Name: Kelly Dhanraj-Singh.

Dataset: Electric Vehicle Population.

Table of Contents

Problem Statement

While electric vehicles have gained popularity in recent years, the lack of accessible clean alternative fuel remains a significant barrier amongst other harmful effects to the environment. Specific models and makes of electric vehicles either offer or lack this alternative. Likewise, the vehicle type also plays a role in whether the vehicles offer this alternative. Resultingly, damage to the environment occurs due to fuel combustion and pollution. With this knowledge, consumers can make informed decisions on which electric vehicle to purchase. Likewise, it can challenge manufacturers to produce only environmentally sustainable electric vehicles.

Problem Analysis

It is noted not all electric vehicles offer clean alternative fuel. The models, makes and types of these electric vehicles can determine if they offer this attribute. Research has been done proving the extent of reliability of these vehicles. Likewise, since the data set was extracted from only the Washington State, research was taken from this area.

The given dataset contains 17 different columns, including: VIN, County, City, State, Postal Code, Model Year, Make, Model, Electric Vehicle Type, Clean Alternative Fuel Vehicle Eligibility, Electric Range, Base MSRP, Legislative District, Vehicle Location, Electric Utility and Census Tract. My project mainly focuses on the following columns: Model Year, Make, Model, Electric Vehicle Type, Clean Alternative Fuel Vehicle Eligibility and Electric Range. The Model Year and Electric Range are considered numerical data, while the other columns of interest are considered categorical data.

To elucidate, the Model Year provides the model year for a given vehicle, the make and model provides the make and model for a given vehicle, whilst the electric vehicle type can either be a battery-electric vehicle or a plug-in hybrid electric vehicle. Additionally, the clean alternative fuel vehicle eligibility column determines if the vehicle offers alternative fuel, and the electric range gives the electric range for a given vehicle. The following paragraphs provide similar research and findings previously done.

According to “Electric Vehicles Are Improving, but Charging and Battery Issues Persist in Consumer Reports’ 2023 Annual Auto Reliability Survey” 2023, electric vehicles provide more issues than internal combustion engine vehicles. Electric vehicles are relatively new and thus it is difficult to detect and fix the problems quickly. Likewise, a survey was done with “over 330,000 vehicles”, models spanning over the years 2000 to 2023. The findings concluded that the “electric vehicles have up to twelve different trouble areas” and specifically the plug-in electric vehicles experience trouble areas encompassing other vehicle problems.

Additionally, Ryan (2022), concluded “Barriers to wider adoption of electric vehicles include a lack of charging stations.” The Interagency Electric Vehicle Coordinating Council in Washington completed a Transportation Electrification Strategy in 2023. “WSDOT – Gray Notebook Electric Vehicles -Public Charging Stations” 2023, this stated there are “291 direct current fast charging station locations” in Washington. Since, there are a limited amount of charging stations, the electric vehicles should have a clean alternative fuel source instead of overloading on one charging station.

Building to this information, according to the WSDOT’s “WA Interactive Electric Vehicle Charging Map” (2023), there is a concentration of charging stations in the west and alarmingly few charging stations in the east of Washington. Resultingly, an overload of electric vehicles in one charging station can lead to damage to the environment.

Additionally, according to the International Energy Agency (2023), multiple electric vehicles on one charging station, “can lead to power quality issues, such as voltage drops”, “this can affect the performance and longevity of the grid infrastructure.”

Moreover, according to McLaren et al. (2016), overloading one on charging station “can lead to increased electricity demand”, “ramping up production, potentially increasing emissions if the energy is sourced from fossil fuels.” Additionally, more batteries will be necessary “due to degradation from overloading.” Hence, more raw materials will have to be sourced, “involves mining and processing activities that can be harmful to the environment.”

Lastly, this project will analyse the similarities between the models, makes and types of electric vehicles and if they offer a clean alternative fuel. The program aims to output the average electric range for a specific make and model combination. To tie in with this, the make and model combination with the highest average electric range should be outputted. Additionally, the electric vehicle type which produces the highest number of “Clean Alternative Fuel Vehicles” should be outputted. As stated previously, the electric vehicle type can either be a battery electric vehicle or a plug-in hybrid electric vehicle. Similarly, the model year with the highest number of “Clean Alternative Fuel Vehicles” produced should be outputted. Finally, the make and model combination which produces the largest amount of “Clean Alternative Fuel Vehicles” should be outputted.

This knowledge can aid both the consumers and manufacturers. The consumers can now make an informed decision on which electric vehicle, they should purchase, based on the make, model, clean alternative fuel vehicle eligibility, model year and average electric range. Whilst manufacturers are challenged with making all electric vehicles environmentally sustainable.

Data Flow

Algorithm for loading from CSV file (data set) into a vector of structs.

1. Include the necessary header files.
2. Create a struct named VehicleData to store the data in each row of the CSV file.
3. Create a vector of type VehicleData structs.
4. Declare a function that converts a string to an integer, making use of the stoi function. If the string is empty, return 0.
5. Declare a function that converts a string to a float, making use of the stof function. If the string is empty, return 0.
6. Declare a void function called LoadDataFromCSVFile.
7. Open the CSV file ("Electric_Vehicle_Population_Data") for reading the data.
8. Check to see if the file was successfully opened. If the file was successfully opened, print a success message to the screen.
9. Else if the file was not opened successfully, print an error message to the screen.
10. Declare a string variable called line.
11. Since the first line of the CSV file contains the headings for the columns, use the getline function to read and discard the first line of the CSV file.
12. Declare a string stream object named ss to temporarily store the 'line' of the CSV file.
13. Declare a string variable named field.
14. Declare a vector of strings of size 17 named row.
15. Declare a while loop such that if the fields are separated by a comma, add the field to the vector row.
16. If any of the columns are missing data, skip over them.
17. Create a VehicleData object and put its corresponding fields.
18. Add the vehicle data to the vector data.
19. Close the input file.
20. Print a success message if the data was loaded correctly and print the number of records loaded.

Algorithm to Calculate the Average Electric Range for a Given Model and Make.

1. Create a function named avgElectricRange, which returns a double value and its' input parameters including a given make and model from the user.
2. Create and initialize an integer variable named sum to zero. This will keep track of the total electric range.
3. Create and initialize an integer variable named count to zero. This will keep track of the number of the specific model and make combination.
4. Create a for loop that iterates through each vehicle in the data vector.
5. If the make and model of the vehicle inputted by the user is equal to the make and model when iterated through the data vector, then the sum of the electric range should increase by the electric range in the data vector.
6. Then increment the count variable by one.
7. If the count is zero, then return zero as well. This means the make and model combination does not exist.
8. Else, return the sum of the electric range divided by the count variable, which is the average electric range.
9. End the function.

Algorithm for Determining the Make and Model Combinations and Which Combination Produces the Highest Average Electric Range.

1. Create a struct called MakeModel that intakes the make and model of vehicles.
2. Create a function named GetMakeModelCombos that returns a vector of MakeModel structures.
3. Declare a vector of MakeModel structures named uniqueMakeModels to store the make and model combinations.
4. Create a for loop to iterate through each vehicle object in the data vector.
5. For each vehicle object, create a MakeModel structure named makeModel that stores its make and model.
6. Declare a Boolean variable named found and initialize it to false.
7. Iterate through the uniqueMakeModelCombos vector to check if the make and model combination already exists. If this combination exists, return true and exit the loop.
8. Else if this combination does not exist, return false and add the make and model to the vector, uniqueMakeModelCombos.
9. Return the uniqueMakeModelCombos vectors.
10. Create a function named detMakeModelWithHighestAvg that returns a MakeModel structure.
11. Call the GetMakeModelCombos function.
12. Create and initialize a double variable named maxAverage to 0.0.
13. Create a MakeModel variable named maxMakeModel to store the make and model combination with the highest average electric range.
14. Iterate through each makeModel combination in the uniqueMakeModelCombos vector.
15. Create a double variable named average to calculate the average electric range for the make and model combination. The avgElectricRange function is utilized in this step.
16. If the calculated average electric range is greater than the maxAverage, then the maxAverage should now be equal to that average electric range value.
17. Else if the maxAverage is greater than the calculated average electric range, then the maxAverage remains the same.
18. This is done for all the make and model combinations in the uniqueMakeModelCombos vector.
19. Return the maxMakeModel which returns the make and model combination that produces the highest average electric range.

Algorithm for the Comparison of Average Electric Ranges.

1.) The user is prompted for the make and model of any vehicle.
2.) The average electric range is calculated for the make and model combination, calling the avgElectricRange function.
3.) The detMakeModelWithHighestAvg function is also called to get the make and model combination that produces the highest average electric range.
4.) The calculated average electric range is then compared with the highest average electric range.
5.) If the average electric range for the make and model combination is less than the highest average electric range, return true.
6.) Else return false.

Algorithm for the Printing of Comparison of Average Electric Ranges.

1.) The average electric range for a given make and model combination is used.
2.) The make and model combination with the highest average electric range is found by calling the detMakeModelWithHighestAvg function.
3.) The average electric range for all make and model combinations are calculated.
4.) The input average electric range is compared with the highest average electric range.
5.) If the input average electric range is less than the highest average electric range, return the string, “The average electric range is less than the highest average electric range.”
6.) Else, return the string, “The average electric range is equal to the highest average electric range.”

Algorithm to Determine the Electric Vehicle Type of the Make and Model Combination.

1.) Create a function named detElectricVehicleType, which returns a string and its’ input parameters are two strings, ie the make and model of the vehicle.
2.) Create a for loop to iterate through the vehicle objects make and model in the data vector.
3.) If the makes and models match, ie from the iteration and from the user’s input, create an if-else if conditional. If the vehicle type is a “Battery Electric Vehicle”, then return BEV.
4.) Else return “PHEV.”
5.) Else output “The electric vehicle type could not be determined.” Return “Unknown Electric Vehicle Type.”
6.) End the function.

Algorithm to Determine which Electric Vehicle Type Produces More Clean Alternative Fuel Vehicles.

1. Create a function named CAFVSByType which returns an integer and its’ input parameters including the vehicle type.
2. Create and initialize an integer variable named count to zero.
3. Create for loop that iterates through each vehicle object in the data vector.
4. If the input vehicle type is equal to the vehicle type when iterated through the data vector and it is “Clean Alternative Fuel Vehicle Eligible”, increment the count variable by one.
5. This is repeated for all in the data vector.
6. Return the final count variable.
7. In the main function, call the CAFVSByType function twice. This is because, in the dataset, there are only two electric vehicle types: Battery Electric Vehicle (BEV) and Plug-in Hybrid Electric Vehicle (PHEV).
8. Output the count values of each electric vehicle type.
9. Compare the count values of each electric vehicle type. Use an if-else conditional to do so. If the count value of the Battery Electric Vehicle (BEV) is greater than that of the Plug-in Hybrid Electric Vehicle (PHEV), then print to the screen that the Battery Electric Vehicles produce a larger amount of clean alternative fuel vehicles.
10. Else if the count value of the Plug-in Hybrid Electric Vehicle (PHEV) is greater than that of the Battery Electric Vehicle (BEV), then print to the screen that the Plug-in Hybrid Electric Vehicles produce a larger amount of clean alternative fuel vehicles.
11. Else if the count values are equal, print to the screen they produce the same amount of clean alternative fuel vehicles.

Algorithm to Determine the Number of Clean Alternative Fuel Vehicles Based on the Model Year.

1. Create a function called CAFVSByYear, which returns an integer and its’ input parameter is the model year.
2. Create and initialize an integer variable named count to zero.
3. Create a for loop to iterate through each vehicle object in the data vector.
4. If the input vehicle’s model year is equal to the model year when iterated through the data vector and it is Clean Alternative Fuel Vehicle Eligible, then increment the count variable by one.
5. Return the count variable.
6. In the main function, call the CAFVSByYear function and input any model year.
7. It will then output the amount of eligible clean alternative fuel vehicles for that model year.

Algorithm for Determining the if the User’s Input Model Year is Valid.

1.) Create a function named detModelYear, that returns an integer.

2.) Create an integer variable named modelYear to store the user’s input.

3.) Create and initialize a Boolean variable named validInput to false.

4.) Create a do-while loop.

5.) Within the loop, prompt the user to input a model year.

6.) Create a string variable named modelYearInput to store the user’s model year.

7.) Use getline function to read the user’s input and convert it to an integer, using the stringToInt function.

8.) If the model year entered falls within the range 1997-2024, then validInput is now set true.

9.) Exit the loop.

10.)If, the model year does not fall within the range 1997-2024, print an error message and prompt the user to enter a valid model year.

11.)The user is prompted until a valid model year is entered.

12.)Return the valid model year.

Algorithm for Determining the Model Year Which Produces the Largest Amount of Eligible Clean Alternative Fuel Vehicles.

1. Create a function named detModelYearWithMaxCAFVS that returns an integer.
2. Create and initialize a vector of integers named counts that has a size of 2025 elements, this includes the years from1997 to 2024.
3. Create a for loop to iterate through each vehicle object in the data vector.
4. If the vehicle is “Clean Alternative Fuel Vehicle Eligible” and its’ model year falls within the range, 1997-2024, then increment the count for that model year, in the counts vector.
5. Create and initialize an integer variable named maxCount to zero. This will store the maximum number of clean alternative fuel eligible vehicles.
6. Create and initialize another integer variable named maxYear. This will store the model year which corresponds to the maximum number of clean alternative fuel eligible vehicles.
7. Iterate through the counts vector to determine the maximum count, ie the maximum number of CAFVS, and its’ corresponding model year.
8. Return the model year with largest number of CAFVS, ie “maxYear.”

Algorithm for Determining the Make with the Largest Number of Clean Alternative Fuel Vehicles.

1.) The function named, “detMakeWithMostCAFVS” returns a string variable, ie it returns the make of the vehicle which produces the largest number of clean alternative fuel vehicles.

2.) Create and initialize a string variable named “maxMake.” This will store the make of the vehicle which produces the largest number of clean alternative fuel vehicles.

3.) Create and initialize an integer variable named “maxCount” to zero. This will track the number of clean alternative fuel vehicles for the “maxMake.”

4.) Iterate through each vehicle object in the data vector using a for loop.

5.) Create and initialize a counter variable named, count to zero.

6.) If the vehicle is a Clean Alternative Fuel Vehicle eligible, increment the counter variable by one.

7.) The “maxMake” will now contain the make with the largest number of CAFVS.

8.) Return the “maxMake.”

Algorithm for Determining the Model with the Largest Number of Clean Alternative Fuel Vehicles.

1.) The function named, “detModelWithMosCAFVS” returns a string variable, ie it returns the model of the vehicle which prodcues the largest number of clean alternative fuel vehicles.

2.) Create and initialize a string variable name “maxModel.” This will store the model of the vehicle which produces the largest number of clean alternative fuel vehicles.

3.) Iterate through each vehicle object in the data vector using a for loop.

4.) Create and initialize a counter variable named, count to zero. This counter variable keeps track of the number of occurrences in which the current model matches with the given make.

5.) If the vehicle is a Clean Alternative Fuel Vehicle eligible, increment the counter variable by one.

6.) The “maxModel” will now contain the model with the largest number of CAFVS.

7.) Return the “maxModel.”

Implementation

```
//Including header files:
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <sstream>

//Creating a struct:
struct VehicleData {
    std::string VIN;
    std::string County;
    std::string City;
    std::string State;
    int  PostalCode;
    int ModelYear;
    std::string Make;
    std::string Model;
    std::string ElectricVehicleType;
    std::string CleanAlternativeFuelVehicleEligibility;
    int ElectricRange;
    int BaseMSRP;
    int LegislativeDistrict;
    float DOLVehicleID;
    std::string VehicleLocation;
    std::string ElectricUtility;
    float Censustract;
};

//Creating a struct that will hold make and model combinations
struct MakeModel {
    std::string Make;
    std::string Model;
};

//Creating a vector of a struct named data:
std::vector<VehicleData> data;

//Function to convert a string to an integer
int stringToInt(const std::string& word) {
    if (word.empty()) {
        return 0;
    }
    else {
        return std::stoi(word);
    }
}

//Function to convert a string to a float
float stringToFloat(const std::string& word) {
    if (word.empty()) {
        return 0.0f;
    }
    else {
        return std::stof(word);
    }
}

//Void function to load the data from the CSV file:
//Reads data from the CSV file and populate the data vector:
void LoadDataFromCSVFile() {
    std::ifstream infile("Electric_Vehicle_Population_Data.csv");
    if (!infile.is_open()) {
        std::cerr << "Error: The file could not be found." << std::endl;
        return;
    }
    else {
        std::cout << "File was found and opened! " << std::endl;
    }

    //Creating a string to obtain lines of data from the CSV file:
    std::string line;

    //Reads and discards the first line of the CSV file:
    std::getline(infile, line); // Read and discard the header line

    //Inserting each field of data into the row vector:
    while (std::getline(infile, line)) {
        std::stringstream ss(line);
        std::string field;
        std::vector<std::string> row;
        while (std::getline(ss, field, ',')) {
            row.push_back(field);
        }

        //Function that allows skipping of incomplete rows:
        if (row.size() != 17) {
            continue;
        }

        //Creating a vehicle object of VehicleData struct type:
        VehicleData vehicle;
        vehicle.VIN = row[0];
        vehicle.County = row[1];
        vehicle.City = row[2];
        vehicle.State = row[3];
        vehicle.PostalCode = stringToInt(row[4]);
        vehicle.ModelYear = stringToInt(row[5]);
        vehicle.Make = row[6];
        vehicle.Model = row[7];
        vehicle.ElectricVehicleType = row[8];
        vehicle.CleanAlternativeFuelVehicleEligibility = row[9];
        vehicle.ElectricRange = stringToInt(row[10]);
        vehicle.BaseMSRP = stringToInt(row[11]);
        vehicle.LegislativeDistrict = stringToInt(row[12]);
        vehicle.DOLVehicleID = stringToFloat(row[13]);
        vehicle.VehicleLocation = row[14];
        vehicle.ElectricUtility = row[15];
        vehicle.Censustract = stringToFloat(row[16]);

        //Adds all the vehicle objects to the data vector:
        data.push_back(vehicle);
    }

    //Closes the input file:
    infile.close();
    std::cout << "Data loaded successfully. The number of records read: "
<< data.size() << std::endl;
}

//Function to calculate the average electric range for a make and model
combination:
//Returns the average electric range for a given make and model
combination:
//If the make and model combination does not exist, an error message is
printed and returns the value 0:
double avgElectricRange(const std::string& make, const std::string& model)
{
    int sum = 0;
    int count = 0;
    for (const auto& vehicle : data) {
        if (vehicle.Make == make && vehicle.Model == model) {
            sum += vehicle.ElectricRange;
            count++;
        }
    }
    if (count == 0) {
        std::cout << "This make and model combination does not exist." <<
std::endl;
        return 0.0;
    }
    else {
        return static_cast<double>(sum) / count;
    }
}

//Creating a vector of MakeModel structure named GetUniqueMakeModels:
//This vector will store each unique make and model combination from the
CSV file:
std::vector<MakeModel> GetMakeModelCombos() {
    std::vector<MakeModel> uniqueMakeModelCombos;
    for (const auto& vehicle : data) {
        MakeModel makeModel = { vehicle.Make, vehicle.Model };
        bool found = false;
        for (const auto& mm : uniqueMakeModelCombos) {
            if (mm.Make == makeModel.Make && mm.Model == makeModel.Model)
{
                found = true;
                break;
            }
        }
        if (!found) {
            uniqueMakeModelCombos.push_back(makeModel);
        }
    }
    return uniqueMakeModelCombos;
}

//Function that returns a MakeModel structure:
//Function returns the make and model that produces the highest average
electric range:
MakeModel detMakeModelWithHighestAvg() {
    auto uniqueMakeModels = GetMakeModelCombos();

    double maxAverage = 0.0;
    MakeModel maxMakeModel;

    for (const auto& makeModel : uniqueMakeModels) {
        double average = avgElectricRange(makeModel.Make,
makeModel.Model);
        if (average > maxAverage) {
            maxAverage = average;
            maxMakeModel = makeModel;
        }
    }

    return maxMakeModel;
}

//Function that determines if the calculated average electric range is
less than or equal to the highest average electric range:
bool avgComparison(const std::string& make, const std::string& model) {
    double averageElectricRange = avgElectricRange(make, model);
    return false;
    MakeModel maxMakeModel = detMakeModelWithHighestAvg();
    double highestAverage = avgElectricRange(maxMakeModel.Make,
maxMakeModel.Model);
    return averageElectricRange < highestAverage;
}


//Function that outputs if the calculated average electric range is less
than or equal to the highest average electric range:
std::string comparingAvgWithHighest(double average) {
    MakeModel maxMakeModel = detMakeModelWithHighestAvg();
    double highestAverage = avgElectricRange(maxMakeModel.Make,
maxMakeModel.Model);
    if (average < highestAverage) {
        return "The average electric range is less than";
    }
    else if (average == highestAverage) {
        return "The average electric range is equal to";
    }
}


//Function that determines if a make and model combination is a BEV or
PHEV:
std::string detElectricVehicleType(const std::string& make, const
std::string& model) {
    for (const auto& vehicle : data) {
        if (vehicle.Make == make && vehicle.Model == model) {
            if (vehicle.ElectricVehicleType == "Battery Electric Vehicle
(BEV)") {
                return "BEV";
            }
            else if (vehicle.ElectricVehicleType == "Plug-in Hybrid
Electric Vehicle (PHEV)") {
                return "PHEV";
            }
        }
    }

    std::cerr << "The electric vehicle type could not be determined." <<
std::endl;
    return "Unknown Electric Vehicle Type";
}


//Function that determines the number of CAFVs for any vehicle type:
int CAFVSByType(const std::string& vehicleType) {
    int count = 0;
    for (const auto& vehicle : data) {
        if (vehicle.ElectricVehicleType == vehicleType &&
vehicle.CleanAlternativeFuelVehicleEligibility == "Clean Alternative Fuel
Vehicle Eligible") {
            count++;
        }
    }
    return count;
}


//Function that determine the number of CAFVs for any model year:
int CAFVSByYear(int modelYear) {
    int count = 0;
    for (const auto& vehicle : data) {
        if (vehicle.ModelYear == modelYear &&
vehicle.CleanAlternativeFuelVehicleEligibility == "Clean Alternative Fuel
Vehicle Eligible") {
            count++;
        }
    }
    return count;
}


//Determines if the model year is valid:
int detModelYear() {
    int modelYear;
    bool validInput = false;
    do {
        std::cout << "Please enter a model year : ";
        std::string modelYearInput;
        std::getline(std::cin, modelYearInput);

        // Convert the input string to an integer
        modelYear = stringToInt(modelYearInput);

        // Check if the model year is within the valid range
        if (modelYear >= 1997 && modelYear <= 2024) {
            validInput = true;
        }
        else {
            std::cout << "Please enter a valid model year." << std::endl;
        }
    } while (!validInput);

    return modelYear;
}


//Function that determines the model year which prodcues the largest
number of CAFVs:
int detModelYearWithMaxCAFVS() {
    std::vector<int> counts(2025, 0);
    bool validModelYearFound = false;
    for (const auto& vehicle : data) {
        if (vehicle.CleanAlternativeFuelVehicleEligibility == "Clean
Alternative Fuel Vehicle Eligible" &&
            vehicle.ModelYear >= 1997 && vehicle.ModelYear <= 2024) {
            counts[vehicle.ModelYear - 1997]++;
            validModelYearFound = true;
        }
    }
    if (!validModelYearFound) {
        std::cerr << "Sorry, an invalid model year was found." <<
std::endl;
        return -1;
    }
    int maxCount = 0;
    int maxYear = 0;
    for (int year = 0; year < counts.size(); ++year) {
        if (counts[year] > maxCount) {
            maxCount = counts[year];
            maxYear = year + 1997;
        }
    }
    return maxYear;
}


// Function to determine the make with the highest number of CAFVs
std::string detMakeWithMostCAFVS() {
    std::string maxMake;
    int maxCount = 0;

    // Iterate through the data vector
    for (const auto& vehicle : data) {
        if (vehicle.CleanAlternativeFuelVehicleEligibility == "Clean
Alternative Fuel Vehicle Eligible") {
            std::string make = vehicle.Make;

            // Count occurrences of the current make
            int count = 0;
            for (const auto& otherVehicle : data) {
                if (otherVehicle.CleanAlternativeFuelVehicleEligibility ==
"Clean Alternative Fuel Vehicle Eligible" &&
                    otherVehicle.Make == make) {
                    ++count;
                }
            }
            // Update maxCount and maxMake if necessary
            if (count > maxCount) {
                maxCount = count;
                maxMake = make;
            }
        }
    }

    return maxMake;
}

// Function to determine the model with the highest number of CAFVs for a
given make
std::string detModelWithMostCAFVS(const std::string& make) {
    std::string maxModel;
    int maxCount = 0;

    // Iterate through the data vector
    for (const auto& vehicle : data) {
        if (vehicle.CleanAlternativeFuelVehicleEligibility == "Clean
Alternative Fuel Vehicle Eligible" &&
            vehicle.Make == make) {
            std::string model = vehicle.Model;

            // Count occurrences of the current model for the given make
            int count = 0;
            for (const auto& otherVehicle : data) {
                if (otherVehicle.CleanAlternativeFuelVehicleEligibility ==
"Clean Alternative Fuel Vehicle Eligible" &&
                    otherVehicle.Make == make && otherVehicle.Model ==
model) {
                    ++count;
                }
            }

            // Update maxCount and maxModel if necessary
            if (count > maxCount) {
                maxCount = count;
                maxModel = model;
            }
        }
    }

    return maxModel;
}

int main() {

    LoadDataFromCSVFile();

    //Prompting for user's input:
    std::string make, model;
    std::cout << "Please enter a make for a vehicle: ";
    std::getline(std::cin, make);
    std::cout << "Please enter a model for a vehicle: ";
    std::getline(std::cin, model);


    //Calculates the average electric range:
    double averageElectricRange = avgElectricRange(make, model);
    std::cout << "Average electric range for make: " << make << " and
model: " << model << " is: " << averageElectricRange << std::endl;
    std::cout << comparingAvgWithHighest(averageElectricRange) << " the
highest average." << std::endl;

    //Prints the make and model combination with the highest average
electric range:
    MakeModel maxMakeModel = detMakeModelWithHighestAvg();
    double highestAverage = avgElectricRange(maxMakeModel.Make, 
maxMakeModel.Model);
    std::cout << "Make with highest average electric range: " << 
maxMakeModel.Make << ", Model: " << maxMakeModel.Model << ", and its 
average electric range is: " << highestAverage << std::endl;


    // Determines the electric vehicle type
    std::string electricVehicleType = detElectricVehicleType(make, model);
    std::cout << "The electric vehicle type is: " << electricVehicleType 
<< std::endl;


    //Determines the number of CAFVS based on the electric vehicle type:
    int bevCount = CAFVSByType("Battery Electric Vehicle (BEV)");
    int phevCount = CAFVSByType("Plug-in Hybrid Electric Vehicle (PHEV)");

    //Prints the number of CAFVS based on the electric vehicle type:
    std::cout << "The number of Clean Alternative Fuel Vehicles for 
Battery Electric Vehicles (BEV): " << bevCount << std::endl;
    std::cout << "The number of Clean Alternative Fuel Vehicles for Plug-
in Hybrid Electric Vehicles (PHEV): " << phevCount << std::endl;


    //Conditional to output which electric vehicle type prodcues more 
CAFVS:
    if (bevCount > phevCount) {
        std::cout << "Battery Electric Vehicles (BEV) produce more Clean 
Alternative Fuel Vehicles." << std::endl;
    }
    else if (bevCount < phevCount) {
        std::cout << "Plug-in Hybrid Electric Vehicles (PHEV) produce more 
Clean Alternative Fuel Vehicles." << std::endl;
    }
    else {
        std::cout << "Both Battery Electric Vehicles (BEV) and Plug-in 
Hybrid Electric Vehicles (PHEV) produce the same number of Clean 
Alternative Fuel Vehicles." << std::endl;
    }

    int modelYear = detModelYear();
    //Prints the number of CAFVS based on the model year:
    int cleanFuelVehiclesCountByYear = CAFVSByYear(modelYear);
    std::cout << "The number of clean alternative fuel vehicles for model 
year " << modelYear << " is: " << cleanFuelVehiclesCountByYear << 
std::endl;

    //Prints the model year with the most CAFVS:
    int maxYear = detModelYearWithMaxCAFVS();
    std::cout << "The model year with the largest production of clean 
alternative fuel vehicles is: " << maxYear << std::endl;

    //Prints the value of the CAFVS based on the model year with the most 
CAFVS:
    int maxYearCount = CAFVSByYear(maxYear);
    std::cout << "The number of clean alternative fuel vehicles for the 
maximum model year " << maxYear << " is: " << maxYearCount << std::endl;

    //Prints the make and model with the largest number of CAFVS:
    std::string makeWithMostCAFVS = detMakeWithMostCAFVS();
    std::string modelWithMostCAFVs = 
detModelWithMostCAFVS(makeWithMostCAFVS);
    std::cout << "The make with the highest number of Clean Alternative 
Fuel Vehicles is: " << makeWithMostCAFVS << std::endl;
    std::cout << "The model associated with this make is: " << 
modelWithMostCAFVs << std::endl;

    return 0;
```

}

Test Cases

As previously stated, the given dataset, CSV file, contains several rows and 17 columns of data. The seventeen columns include: VIN, County, City, State, Postal Code, Model Year, Make, Model, Electric Vehicle Type, Clean Alternative Fuel Vehicle Eligibility, Electric Range, Base MSRP, Legislative District, Vehicle Location, Electric Utility and Census Tract.

Within the code, a struct called "VehicleData" is created and stores each row of data from the dataset. Another struct called "MakeModel" is created to be utilized in the "GetMakeModelCombos" function. A vector of VehicleData called "data" is used to store the vehicle data from CSV file.

The code then defines "stringToInt" and "stringtoFloat" functions to convert string variables to integers and floats respectively. This is done to facilitate the different data types within the CSV file, so that the data can be read correctly.

The "LoadDataFromCSVFile" is a void function that loads the data from the CSV file. Initially, it uses an ifstream object to open the CSV file and prints an error message if the file cannot be found. It also uses getline to obtain each line of data from the CSV file. The first line of the data is the headings of the columns, thus it is read and discarded.

The function then uses a stringstream object to parse each line of data and temporarily store each field of data. Once a line is read, its' split into individual fields, separated by a comma. Each field is then pushed into a vector row. The function then makes a check to see if the row contains less than the expected number of columns, ie 17. These incomplete rows are skipped, and the process continues.

The VehicleData struct is then created and populated with the corresponding fields. The string fields are assigned without any changes, whilst the integer and float fields utilize the stringToInt and stringToFloat functions. This process continues until all lines of the CSV file have been passed and processed. When all the data is loaded, the function finally closes the ifstream object and prints a success message of loading the data and the number of records loaded.

The "avgElectricRange" function returns a double variable, ie it is supposed to return the average electric range, and its' input parameters include two string variables, the make and model of the vehicle. Two integer variables named count and sum are created and initialized to zero. The sum variable keeps track of the total electric range for the make and model combination. On the other hand, the count variable keeps track of the occurrences of the make and model combination. A for loop is initiated where it iterates through the data vector for each vehicle object. An if conditional is used, if the make of the vehicles matches the user's input for the make and model conditional, then the sum is now updated. Likewise, the count should increment by one. Now, out of the loop, an if-else conditional is utilized, the function checks if the count is zero, ie the make and model combination the user inputted does not exist, then 0.0 is returned. Else, the average is calculated by dividing the sum variable by the count variable. A type cast is used to ensure the data types match.

Another function called, “GetMakeModelCombos” returns a vector of the struct MakeModel, named uniqueMakeModelCombos. This function extracts and store the unique make and model combinations of vehicles in the dataset.

Within this function, the vector, “uniqueMakeModelCombos” is initialized to store these make and model combinations for each vehicle. It iterates through each vehicle object in the data vector. For each vehicle object, a MakeModel struct is created to store the corresponding make and model combination. A Boolean variable, called “found” is created and initialized to false.

Another for loop is created and used to iterate through the uniqueMakeModelCombos vector. Each MakeModel combination is compared with the current make and model combination. If they are both equal, then the found variable is set to true. This is because the make and model combination already exist in the uniqueMakeModelCombos vector. If the make and model combination does not already exist in the uniqueMakeModelCombos vector, then the combination is added to the vector using the “push_back” method. Finally, when all the vehicle objects have been processed, the function returns the uniqueMakeModelCombos vectors.

The function named, “detMakeModelWithHighestAvg” returns a MakeModel struct. It aims to identify the make and model that produces the highest average electric range. Initially, the “GetMakeModelCombos” function is called to obtain a vector of unique MakeModel combinations from the dataset. A double variable named “maxAverage” is created and initialized to zero. This is used to store the maximum average electric range. The maxMakeModel is also created and used to store the make and model combination corresponding to the maximum average electric range. The function iterates through each make and model combination from “GetMakeModelCombos.” For each make and model combination, the average electric range is calculated using the “avgElectricRange” function. An if conditional is used to compare if the average calculated is greater than the current maximum average in the “maxAverage” variable. If this is true, then the “maxAverage” variable is now updated. When iterated through all the make and model combinations, the function returns a MakeModel struct named, “maxMakeModel.” This contains the make and model combination with the highest average electric range.

Moreover, the avgComparison function determines if the make and model combination produce an electric range less than or equal to the highest average electric range. The comparingAvgWithHighest function prints if this result.

The detElectricVehicleType function aims to determine whether a given make and model combination corresponds to a Batter Electric Vehicle or a Plug-in Hybrid Electric Vehicle. It iterates through the data vector to find the input make and model. If they match, it returns either “BEV” or “PHEV” depending on the make and model. If the make and model combination does not exist, it prints an error message.

The “CAFVSByType” function operates similarly. It returns an integer storing the count of vehicles that are clean alternative fuel vehicle eligible. An integer variable named, “count” is created and initialized to zero. It iterated through each vehicle object in the data vector. The function checks if the electric vehicle type matches the inputted electric vehicle type and if it is clean alternative fuel vehicle eligible. If these conditions are true, the count variable is incremented by one. The function finally returns the final count value of the number of eligible clean alternative fuel vehicles for the specified electric vehicle type.

The "CAFVSByYear" function returns the count of vehicles manufactured in a specified year that are eligible as clean alternative fuel vehicles. It initializes a count variable to zero and iterates through each vehicle in the dataset, incrementing the count if the vehicle's model year matches the input and it's eligible as a clean alternative fuel vehicle. Finally, it returns the count of eligible vehicles for the specified year.

The “detModelYearWithMaxCAFVS” function determines the model year with the highest number of clean alternative fuel vehicles. It checks if the model year falls within the range 1997-2024, firstly and displays an error message if it is invalid. It creates and initializes a vector named counts to store the number of clean alternative fuel vehicles for each model year. It iterates through each vehicle object in the data vector. It increments the count by one if the vehicle is “Clean Alternative Fuel Vehicle Eligible.” Two integer variables, called “maxCount” and “maxYear” are then created and initialized to zero. After iterating through the vehicles and populating the counts vector with the number of clean alternative eligible fuel vehicles for each model year, the function then finds the model year with the highest count of clean alternative eligible fuel vehicles. It compares each count with the current maxCount and if the count is greater than the maxCount, the maxCount variable is updated as well as the maxYear, which stores the model year with the highest number of “Clean Alternative Fuel Vehicles Eligible.” The function then returns the model year with the highest count of clean alternative fuel vehicles.

The detMakeWithMostCAFVS function prints the make which produces the largest number of CAFVS. The variable maxMake serves as a placeholder to store this make. When the function iterates through the data vector, it compares the number of CAFVS of each make with the current maximum value, ie in maxCount. If a make is found with a higher value than maxCount, maxCount is updated and the maxMake is also updated. The maxMake is then returned after iterating through all the makes.

The detModelWithMostCAFVS function operates similarly. In this case, maxModel stores the model which produces the largest number of CAFVS.

These functions are also called in main for the user’s input. This thus tests the code’s functionality and if it handles invalid input. This is necessary for code improvement and testing.

Functionality of Test Cases

Table 1: Displaying Output for Functions

Function Name	Output	
	Success	Error
LoadDataFromCSVFile	File was found and opened!	Error: The file could not be found.
detMakeModelWithHighestAvg	The make with the highest average electric range is: HYUNDAI, and model is: KONA, and its' average electric range is: 258	

CAFVSByType	The number of Clean Alternative Fuel Vehicles for Battery Electric Vehicles (BEV):
	The number of Clean Alternative Fuel Vehicles for Batter Electric Vehicles (BEV):
	(put which one is greater)	

detMakeWithMostCAFVS	TESLA	
detModelWithMostCAFVS	MODEL 3	
detModelYearWithMaxCAFVS	The model year with the largest production of clean alternative fuel vehicles is: 2018	

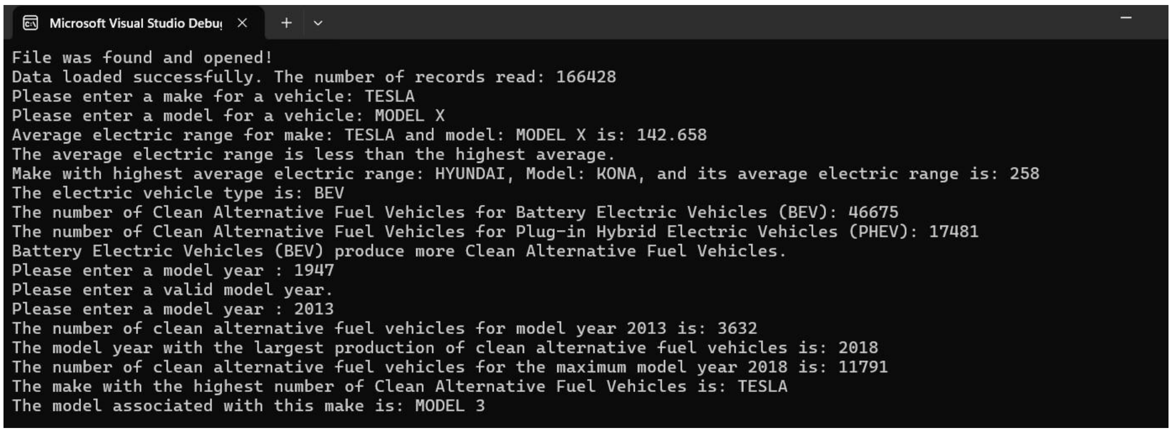
Table 2: Displaying Output for Respective Functions for the Model Year

Function Name	User's Input	Output	
		Success	Error
CAFVSByYear	1947		Please enter a valid model year.
	2010	23	
	2018	11791	
	2023	4366	

Table 3: Displaying Output for Functions for Respective Make and Model Combinations

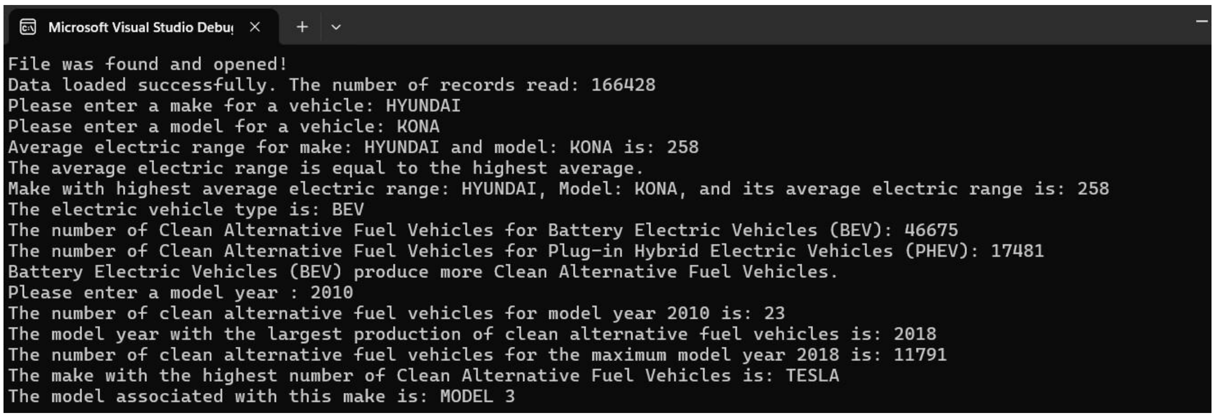
Function Name	User's Input		Output	
	Make	Model	Success	Error
avgElectricRange	TESLA	MODEL X	142.658	
	HYUNDAI	KONA	258	
	NISSAN	LEAF	85.0589	
	TESLA	CYBERTRUCK		0
detElectricVehicleType	TESLA	MODEL X	BEV	
	HYUNDAI	KONA	BEV	
	NISSAN	LEAF	BEV	
	TESLA	CYBERTRUCK		Unknown electric vehicle type. This make and model combination does not exist.
ComparingAvgWithHighest	TESLA	MODEL X	The average electric range is less than the highest average	
	HYUNDAI	KONA	The average electric range is equal to the highest average	
	NISSAN	LEAF	The average electric range is less than the highest average	
	TESLA	CYBERTRUCK		The average electric range is less than the highest average

Output



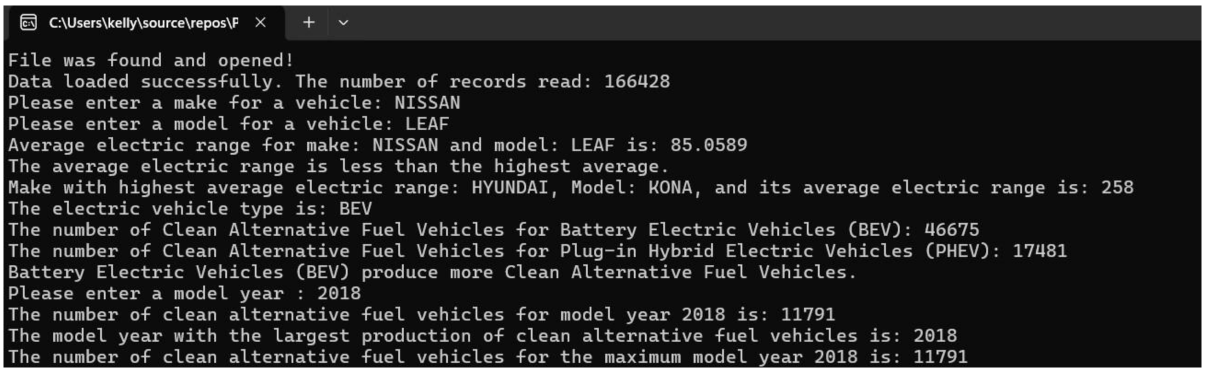
```
File was found and opened!
Data loaded successfully. The number of records read: 166428
Please enter a make for a vehicle: TESLA
Please enter a model for a vehicle: MODEL X
Average electric range for make: TESLA and model: MODEL X is: 142.658
The average electric range is less than the highest average.
Make with highest average electric range: HYUNDAI, Model: KONA, and its average electric range is: 258
The electric vehicle type is: BEV
The number of Clean Alternative Fuel Vehicles for Battery Electric Vehicles (BEV): 46675
The number of Clean Alternative Fuel Vehicles for Plug-in Hybrid Electric Vehicles (PHEV): 17481
Battery Electric Vehicles (BEV) produce more Clean Alternative Fuel Vehicles.
Please enter a model year : 1947
Please enter a valid model year.
Please enter a model year : 2013
The number of clean alternative fuel vehicles for model year 2013 is: 3632
The model year with the largest production of clean alternative fuel vehicles is: 2018
The number of clean alternative fuel vehicles for the maximum model year 2018 is: 11791
The make with the highest number of Clean Alternative Fuel Vehicles is: TESLA
The model associated with this make is: MODEL 3
```

Figure 1:Output for TESLA MODELX, Model Years 1947 and 2013 and Make and Model With Highest CAFVS



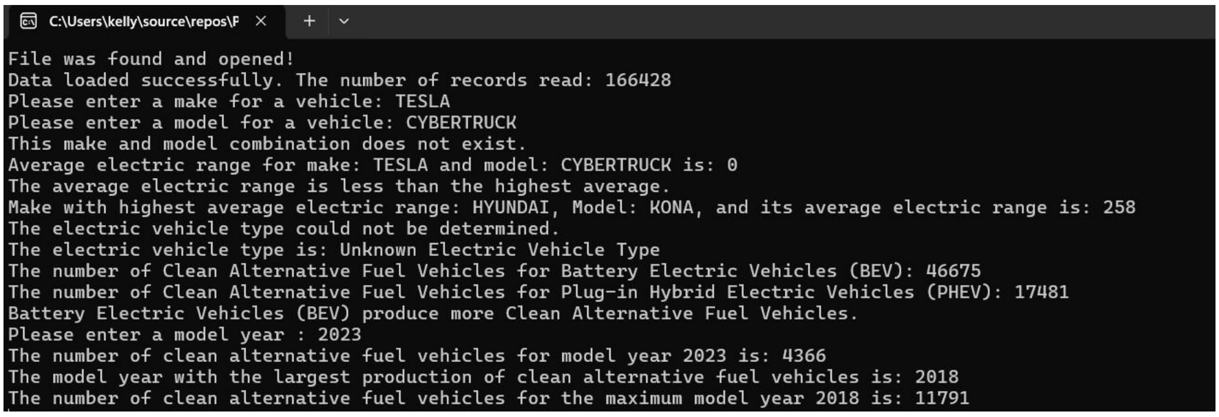
```
File was found and opened!
Data loaded successfully. The number of records read: 166428
Please enter a make for a vehicle: HYUNDAI
Please enter a model for a vehicle: KONA
Average electric range for make: HYUNDAI and model: KONA is: 258
The average electric range is equal to the highest average.
Make with highest average electric range: HYUNDAI, Model: KONA, and its average electric range is: 258
The electric vehicle type is: BEV
The number of Clean Alternative Fuel Vehicles for Battery Electric Vehicles (BEV): 46675
The number of Clean Alternative Fuel Vehicles for Plug-in Hybrid Electric Vehicles (PHEV): 17481
Battery Electric Vehicles (BEV) produce more Clean Alternative Fuel Vehicles.
Please enter a model year : 2010
The number of clean alternative fuel vehicles for model year 2010 is: 23
The model year with the largest production of clean alternative fuel vehicles is: 2018
The number of clean alternative fuel vehicles for the maximum model year 2018 is: 11791
The make with the highest number of Clean Alternative Fuel Vehicles is: TESLA
The model associated with this make is: MODEL 3
```

Figure 2:Output for HYUNDAI KONA, Model Year 2010 and Make and Model With Highest CAFVS



```
File was found and opened!
Data loaded successfully. The number of records read: 166428
Please enter a make for a vehicle: NISSAN
Please enter a model for a vehicle: LEAF
Average electric range for make: NISSAN and model: LEAF is: 85.0589
The average electric range is less than the highest average.
Make with highest average electric range: HYUNDAI, Model: KONA, and its average electric range is: 258
The electric vehicle type is: BEV
The number of Clean Alternative Fuel Vehicles for Battery Electric Vehicles (BEV): 46675
The number of Clean Alternative Fuel Vehicles for Plug-in Hybrid Electric Vehicles (PHEV): 17481
Battery Electric Vehicles (BEV) produce more Clean Alternative Fuel Vehicles.
Please enter a model year : 2018
The number of clean alternative fuel vehicles for model year 2018 is: 11791
The model year with the largest production of clean alternative fuel vehicles is: 2018
The number of clean alternative fuel vehicles for the maximum model year 2018 is: 11791
```

Figure 3:Output for NISSAN LEAF and Model Year, 2018



```
File was found and opened!
Data loaded successfully. The number of records read: 166428
Please enter a make for a vehicle: TESLA
Please enter a model for a vehicle: CYBERTRUCK
This make and model combination does not exist.
Average electric range for make: TESLA and model: CYBERTRUCK is: 0
The average electric range is less than the highest average.
Make with highest average electric range: HYUNDAI, Model: KONA, and its average electric range is: 258
The electric vehicle type could not be determined.
The electric vehicle type is: Unknown Electric Vehicle Type
The number of Clean Alternative Fuel Vehicles for Battery Electric Vehicles (BEV): 46675
The number of Clean Alternative Fuel Vehicles for Plug-in Hybrid Electric Vehicles (PHEV): 17481
Battery Electric Vehicles (BEV) produce more Clean Alternative Fuel Vehicles.
Please enter a model year : 2023
The number of clean alternative fuel vehicles for model year 2023 is: 4366
The model year with the largest production of clean alternative fuel vehicles is: 2018
The number of clean alternative fuel vehicles for the maximum model year 2018 is: 11791
```

Figure 4:Output for TESLA CYBERTRUCK and Model Year 2023

Discussion

Firstly, the loadDatFromCSVFile() function was successfully opened indicated by a success message on the screen. The make and model which produces the highest number of Clean Alternative Fuel Vehicles is TESLA MODEL 3.

However, the highest average electric range, 258, was produced by HYUNDAI, KONA. This make and model is a Battery Electric Vehicle. This make and model is also a Clean Alternative Fuel Vehicle. Hence, this is a potential good choice for consumers in Washington, as it is environmentally sustainable, and it has an extensive driving range.

It was found that Battery Electric Vehicles produce the greater number of Clean Alternative Fuel Vehicles. Hence, a BEV would also be a good indicator of which vehicle to purchase. Additionally, 2018 was the model year which produced the greatest number of Clean Alternative Fuel Vehicles. Therefore, choosing a 2018 model year vehicle would also be a good choice.

With any dataset, there are limitations. One including, this dataset was only based in Washington, hence there were limited reading resources on this specific dataset. Also, more makes and models should have been provided to obtain a better understanding of the data. To improve, my solution, a corelation could have been made between the cities in Washington and the makes and models of the vehicles. An output of various make and model combinations along with their cities can help manufacturers increase their target audience.

Lastly, one recommendation is to encompass the dataset over a larger geographical area to obtain better and more accurate results. A larger geographical area would validate the findings as more data will be provided.

Conclusion

The make and model which is the most environmentally friendly and possess an extensive driving range is HYUNDAI, KONA. The model year which produced the highest number of Clean Alternative Fuel Vehicles is 2018. Additionally, battery Electric Vehicles produced more Clean Alternative Fuel Vehicles, when compared with Plug-in Hybrid Electric Vehicles, making it a better choice for environmentally conscious consumers in Washington.

References

"Electric Vehicles Are Improving, but Charging and Battery Issues Persist in Consumer Reports' 2023 Annual Auto Reliability Survey." 2023. Consumer Reports. November 9, 2023. https://www.consumerreports.org/media-room/press-releases/2023/11/electric-vehicles-are-improving-but-charging-and-battery-issues-persist-in-consumer-reports-2023-annual-auto-reliability-survey/.

International Energy Agency. 2023. "Trends in Charging Infrastructure – Global EV Outlook 2023 – Analysis." IEA. 2023. https://www.iea.org/reports/global-ev-outlook-2023/trends-in-charging-infrastructure.

McLaren, Joyce, John Miller, Eric O'Shaughnessy, Eric Wood, and Evan Shapiro. 2016. "CO2 Emissions Associated with Electric Vehicle Charging: The Impact of Electricity Generation Mix, Charging Infrastructure Availability and Vehicle Type." *The Electricity Journal* 29 (5): 72–88. https://doi.org/10.1016/j.tej.2016.06.005.

Ryan, John. 2022. "Electric Vehicles Have Surged in Washington State. But Gas Cars Still Dominate." Kuow.org. KUOW Public Radio. March 19, 2022. https://www.kuow.org/stories/electric-vehicle-sales-have-surged-in-wa-but-gas-cars-still-dominate.

"WSDOT. "WA Interactive Electric Vehicle Charging Map." 2023 https://wsdot.maps.arcgis.com/apps/CrowdsourceReporter/index.html?appid=6d1e12ec58f842cbaf1b83e3d60e0f09.

"WSDOT - Gray Notebook Electric Vehicles - Public Charging Stations." 2023. Wsdot.wa.gov. 2023. https://wsdot.wa.gov/about/data/gray-notebook/gnbhome/environment/electricvehicles/publicchargingstations.htm.