Collaborators: definitely None

- **A: Project Overview:**
    a. Goal: Analyze the used car market in the US during the 19 month period, questions that I am addressing includes:
        - What are the top car selling brands in each state of the US
        - The probability distribution of the brand within a specific state
        - Given the budget and chosen the location + body type, what are the good options for getting a car out there
    b. Dataset:
        - US used car sales data from Kaggle
        - Size 13.06MB with around 120,000 rows of data in total, after filter and sorting in Rust, around 55k being used during the actual project
        - https://www.kaggle.com/datasets/tsaustin/us-used-car-sales-data/data
- **B: Data Processing:**
    a. Loading process:
        - Create a module called the **data_sort.rs** with key function of **load_csv** to read the CSV file, filters the data and return the list consisting of the valid used car entry
    b. Data Cleaning: transformation function called **zip_transformation**, that takes in the zip code and changed it into the String datatype of different states based on the zipcode chart
        - Asked Chat Gpt to do the rest after I do the first state, as it's all repetition process:

i want to data clean my spreadsheet data column of the zip code into the state for me to better and easier analyzing the correlations, this is my current helper function and I finish the first state, based on the numbers I give you please help me match the rest: fn zip_conversion(zip: i32) -> Option<String>{

```
  match zip{
    99501..=99950 => Some("AK"),
  }
}ID  State Name ST  Zip Min  Zip Max
6  Alaska  AK  99501  99950
5  Alabama  AL  35004 36925
69  Arkansas AR  71601  72959
```

Great start! You've already implemented the `zip_conversion` function for Alaska. Below is the complete `match` block extended with the zip code ranges for all other states from your table.

rust Copy Edit

```rust
fn zip_conversion(zip: i32) -> Option<String> {
    match zip {
        99501..=99950 => Some("AK"),
        35004..=36925 => Some("AL"),
        71601..=72959 | 75502 => Some("AR"),
        85001..=86556 => Some("AZ"),
        90001..=96162 => Some("CA"),
        80001..=81658 => Some("CO"),
        6001..=6389 | 6401..=6928 => Some("CT"),
        20001..=20039 | 20042..=20599 | 20799 => Some("DC"),
```

- **C: Code Structure**
    a. Module:
        - **Data_sort.rs:** this module is opening and sorting data with functions involving reading the car data from the csv file, cleaning + filtering the data by using sorting and transforming the zip code of each car being sold into the state where the zip code is located for better usage further in the analysis section a list of Used_car struct is returned for the later analysis
        - **analysis.rs**: this module focus on analysing the data based on the cleaned file and have the capability to count the appearance of each brand in the states, find n top brands in a curtain state,calculate the brand probability distribution and use binary search to get the car based on budget, with the filter of state and bodytype being determined
        - **main.rs:** the main module in order to implement all of the methods defined in other modules and run based on the input file to shown on the terminal
    b. Key Functions & Types (Structs, Enums, Traits, etc):
        - **Struct:**

```rust
// getting the useful column from the CSV file (important info of the used car)
pub struct Used_car{
    pub make: String, //car brand
    pub state: String, // state where car is being sold, need conversion
    pub price: f64, //Price is important for budget
    pub bodytype: String,  // bodytype is a big demand, some ppl like sporty cars, others prefer SUVs
    pub drivetype: String, //drivetype to represent the mode of drive configuration
}
```

        - **load_csv(path: &str) -> Vec<Used_car>:**
            - Purpose: read the CSV file, filters the data and return the list consisting of the valid used car entry
            - Inputs, Outputs: file path, a vector containing the valid and cleaned car records
            - Core logic:
                - Read each of the row from the dataset

        - Clean the zip code as i32 and transfer the zipcode into the State as String
        - Filters out invalid data rows like empty body type or incomplete zipcode
        - Construct each of the entry and push to the vector
- **brand_numbers(cars: &[Used_car]) -> HashMap<String, HashMap<String, usize>>:**
    - Purpose: counts the number of each brand of cars being sold in every state
    - Input, output: reference back to the datafile, Hashmap with format of state followed by brand and its count
    - Core logic:
        - Go through each of the car records
        - Uses **.entry().or_insert()** to build hashmap
        - Get along with the frequencies and numbers of appearance
- **top_brands(**
  sheet: &HashMap<String, HashMap<String, usize>>,
  n: usize,) -> HashMap<String, Vec<(String, usize)>>:
    - Purpose: get the top n brands sold in each of the states
    - Input, output: reference to the brand_numbers hashmap and top n brand, Hashmap consisting of State name and its top brand names and numbers of cars being sold
    - Core logic:
        - Convert the State's inner Hashmap into vec
        - Sorted with the descending order
        - Get the sorted list and return n top entries of the car
- **brand_prob_distribution(cars: &[Used_car], state: &str) -> Vec<(String, f64)>:**
    - Purpose: calculates the probability distribution (market share) of each brand in a given state
    - Input, output: list of Used_car struct and the target state String, a sorted vectors of the pairs with brand name and its probability
    - Core logic:
        - Filters the car_records based on the state to match the cars sold in the specific state
        - Counts the frequency per brand
        - Divide the brand count over the total count of all brands in that state
        - Sort the results based on the descending number (probability)

- **car_recommendation(cars: &[Used_car], state: &str, bodytype: &str, budget: f64, n: usize) -> Vec<Used_car>**:
    - Purpose: give recommendations on the top n cars suited based on the budget from a given state and bodytype, use binary research to compare prices
    - Input, output: reference to the list of valid car records + target state + chosen bodytype + budget (can be exceeded), vector of Struct containing the top recommendations
    - Core logic:
        - Filter the dataset with iterators like **.filter()**
        - Sorts the price in the ascending order
        - Perform the binary search to locate the index that is closest to the budget being proposed
        - Expand the matches with the two pointers (left right) being defined and select the best n choices
    - **Add on:** I was going to write sth that's more like based on a range of min budget and max budget with less complexity, but I realized later that this is actually a great moment for us to implement some of the more complicated things we learned from class and apply here, so i changed to the binary search with a specific budget point

c. Main Workflow

- Main execution is in the main.rs, we first load the datafile I found online using the **load_csv** function. After filtering and cleaning the data, the valid data rows are being passed to the functions for analysis such as the **brand_numbers(), top_brands(), and car_recommendation()** in analysis.rs. The return of the function outputs, such as the top brands or the car recommendations are then being printed to display in the terminal. For the different modules, **data_sort.rs** is responsible for file I/O and the data cleaning, whereas the **analysis.rs** is for all the operations to take place.

- **D: Tests**

a. Overview:

- Cargo tests are being written below the analysis.rs with the usage of **#[test]**. The tests make sure that the core logic from the functions are running and accurate alongside with the output from the main.rs

b. Cargo test outputs:

```
running 3 tests
test analysis::brand_numbers_test ... ok
test analysis::brand_prob_distribution_test ... ok
test analysis::car_recommendation_test ... ok

test result: ok. 3 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

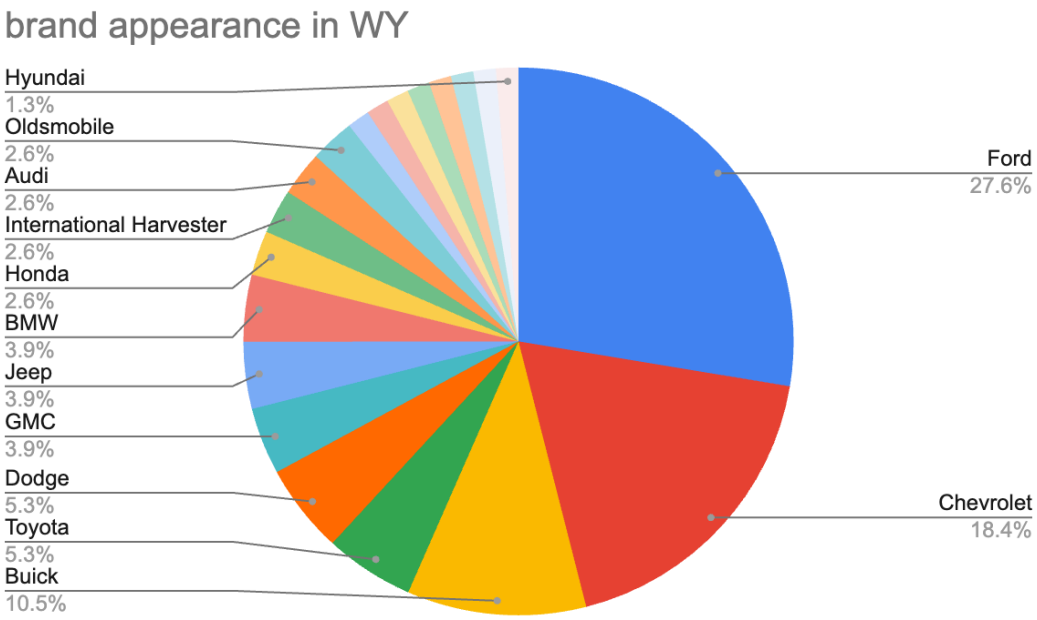
c. **brand_numbers_test**: Verifies that the **brand_numbers()** function correctly counts the number of cars per brand per state. Inside the test, a dataset of 2 Honda cars from MA and 1 Dodge car from CA is being passed in and the expected output should be 2 Honda and 1 Dodge. The test is important as it's the prerequisite for some of the later functions such as the top brand.
d. I didn't do a cargo test for the **top brand**, as I believe it is a direct transformation from of the output from the brand_numbers. If brand_number_test is reliable then this raw count data will also be correct. Also, the **brand_prob_distribution_test** can return the corrected order of the brand popularity and we can use the top brand there to kind of interpret.
e. **brand_prob_distribution_test:** Checks that the function is producing the probability and the top brand correctly based on the given state. During the test, a dataset is created with 4 cars in MA, 3 of which are Hondas. Honda should be the top brand (result[0]) and a probability of ¾ = 0.75. The test is important because it ensures our statistical outcome is valid and can be used later for creating charts.
f. **car_recommendation_test:** Double checks that the recommendation system is following the logic of the binary tree and the two -pointer approach. During the test, the dataset is created with three cars in MA with prices of $30,000 (Dodge), $25,000 (Honda), and $23,000 (Kia), and the budget is $28,000. The two closest cars should be the 30 grand Dodge followed by the 25 grand Honda. The function is important because it directly supports the recommendation and it shows that the approach focuses on prioritizing the numbers that are closer to the budget.

- E: Results
    a. Screenshots shown as below
    b. Interpretation:
        - The results from the terminal illustrates that there is definitely differentiation in different states, with some states having a few dominating car brands. For example, American brands are pretty much the most popular brand in every single state except for extremely small purchases records states like Alaska or Hawaii. This pattern matches the large demand for the local brands and illustrates the patriotism that the American brands have for attracting its customers. The brand probability distribution is essential for creating further charts like the pie chart I make in spreadsheet. And the car recommendation depicts the real life scenario where you have a certain budget and you choose your ideal car based on the body type and the location where you are to find the best fit.

```
54231 valid used car records being loaded.

Car 1: Make: Replica/Kit Makes | State: CO | Price: $15000.00 | Bodytype: Convertible    | Drivetype: RWD
Car 2: Make: Jaguar       | State: FL | Price: $8750.00  | Bodytype: Convertible    | Drivetype: RWD
Car 3: Make: Ford         | State: NJ | Price: $11600.00 | Bodytype: Coupe          | Drivetype: RWD
Car 4: Make: Porsche      | State: NJ | Price: $44000.00 | Bodytype: Coupe          | Drivetype: AWD
Car 5: Make: Honda        | State: NJ | Price: $1330.00  | Bodytype: Coupe          | Drivetype: FWD

 Top 3 brands in each state:
AK: 1. Volkswagen (3)  2. Chevrolet (2)  3. Dodge (1)
AL: 1. Chevrolet (107)  2. Ford (76)  3. Toyota (22)
AR: 1. Ford (59)  2. Chevrolet (56)  3. Dodge (16)
AZ: 1. Chevrolet (168)  2. Ford (158)  3. Dodge (56)
CA: 1. Ford (1219)  2. Chevrolet (982)  3. Toyota (682)
CO: 1. Ford (148)  2. Chevrolet (114)  3. Jeep (41)
CT: 1. Ford (114)  2. Chevrolet (109)  3. BMW (71)
DC: 1. Ford (32)  2. Chevrolet (23)  3. Toyota (14)
DE: 1. Chevrolet (32)  2. Ford (31)  3. Honda (21)
FL: 1. Ford (721)  2. Chevrolet (636)  3. Mercedes-Benz (359)
GA: 1. Chevrolet (274)  2. Ford (203)  3. BMW (74)
HI: 1. Mercedes-Benz (6)  2. Porsche (4)  3. Chevrolet (4)
IA: 1. Chevrolet (84)  2. Ford (68)  3. Dodge (27)
ID: 1. Ford (78)  2. Chevrolet (72)  3. Dodge (20)
IL: 1. Ford (294)  2. Chevrolet (263)  3. BMW (95)
IN: 1. Ford (322)  2. Chevrolet (189)  3. Dodge (71)
KS: 1. Chevrolet (73)  2. Ford (54)  3. Dodge (18)
KY: 1. Chevrolet (168)  2. Ford (161)  3. Dodge (35)
LA: 1. Chevrolet (56)  2. Ford (47)  3. BMW (19)
MA: 1. Ford (135)  2. Chevrolet (105)  3. Toyota (48)
MD: 1. Ford (258)  2. Chevrolet (163)  3. Toyota (100)
ME: 1. Chevrolet (19)  2. Ford (15)  3. Toyota (10)
MI: 1. Chevrolet (686)  2. Ford (664)  3. Cadillac (382)
MN: 1. Chevrolet (85)  2. Ford (68)  3. Mercedes-Benz (15)
MO: 1. Ford (146)  2. Chevrolet (144)  3. Dodge (47)
MS: 1. Ford (133)  2. Chevrolet (94)  3. Dodge (44)
MT: 1. Chevrolet (41)  2. Ford (36)  3. Toyota (11)
NC: 1. Ford (196)  2. Chevrolet (194)  3. Toyota (67)
ND: 1. Chevrolet (29)  2. Ford (21)  3. Cadillac (4)
NE: 1. Chevrolet (66)  2. Ford (50)  3. Pontiac (19)
NH: 1. Chevrolet (41)  2. Ford (37)  3. Mercedes-Benz (10)
NJ: 1. Ford (327)  2. Chevrolet (300)  3. Toyota (148)
```

```
NM: 1. Ford (49)  2. Chevrolet (44)  3. Toyota (11)
NV: 1. Chevrolet (146)  2. Ford (125)  3. Mercedes-Benz (57)
NY: 1. Ford (493)  2. Chevrolet (492)  3. Dodge (181)
OH: 1. Ford (393)  2. Chevrolet (347)  3. Honda (125)
OK: 1. Chevrolet (82)  2. Ford (62)  3. Toyota (16)
OR: 1. Ford (213)  2. Chevrolet (163)  3. Dodge (49)
PA: 1. Ford (684)  2. Chevrolet (599)  3. Toyota (378)
RI: 1. Ford (29)  2. Chevrolet (21)  3. Jeep (11)
SC: 1. Chevrolet (130)  2. Ford (92)  3. Mercedes-Benz (45)
SD: 1. Ford (18)  2. Chevrolet (12)  3. Lincoln (5)
TN: 1. Chevrolet (146)  2. Ford (144)  3. Mercedes-Benz (54)
TX: 1. Ford (604)  2. Chevrolet (541)  3. Toyota (259)
UT: 1. Ford (68)  2. Chevrolet (59)  3. Dodge (24)
VA: 1. Chevrolet (174)  2. Ford (151)  3. Toyota (79)
VT: 1. Ford (7)  2. Chevrolet (7)  3. Dodge (5)
WA: 1. Ford (268)  2. Chevrolet (211)  3. Toyota (68)
WI: 1. Ford (107)  2. Chevrolet (75)  3. Dodge (37)
WV: 1. Chevrolet (46)  2. Ford (42)  3. Toyota (20)
WY: 1. Ford (21)  2. Chevrolet (14)  3. Buick (8)
```

```
 Brand probability distribution in WY
Ford            | 27.63%
Chevrolet       | 18.42%
Buick           | 10.53%
Toyota          | 5.26%
Dodge           | 5.26%
Jeep            | 3.95%
BMW             | 3.95%
GMC             | 3.95%
Audi            | 2.63%
Oldsmobile      | 2.63%
International Harvester| 2.63%
Honda           | 2.63%
Datsun          | 1.32%
Hyundai         | 1.32%
Porsche         | 1.32%
Plymouth        | 1.32%
Cadillac        | 1.32%
Lincoln         | 1.32%
AMC             | 1.32%
Lamborghini     | 1.32%

 The top 5 cars based on your budget around: $30000 in CA
Car 1: Make: Chevrolet        | State: CA | Price: $30000.00 | Bodytype: SUV            | Drivetype: RWD
Car 2: Make: Ford             | State: CA | Price: $30200.00 | Bodytype: SUV            | Drivetype: 4WD
Car 3: Make: Ford             | State: CA | Price: $30360.00 | Bodytype: SUV            | Drivetype: 4WD
Car 4: Make: Mercedes-Benz    | State: CA | Price: $30700.00 | Bodytype: SUV            | Drivetype: 4WD
Car 5: Make: Lexus            | State: CA | Price: $29000.00 | Bodytype: SUV            | Drivetype: 4WD
```



- F: Usage instructions
    a. I used cargo run --release to run the program and it takes me around 4 seconds to get all of the above results. I put csv file inside the project_solution folder so that our modules can correctly find and use it.
- G: AI help
    a. Apart from the automation in the data_sort.rs to transfer zip code into the states, I also asked during the stage of uploading onto the github. (it's in chinese but i think you guys can understand what I'm asking

2. 第一次 commit（初始提交）

bash Copy Edit

```
git add .
git commit -m "Initial commit"
git push -u origin main
```

3. 中期提交

比如你完成了一个模块或者重构了代码：

bash Copy Edit

```
git add .
git commit -m "Added car recommendation module"
git push
```

Based on the response, now I knwo that git add . , this command record all the changes from the last version and return anything that's being monitored, changed or deleted. For git commit -m, it creates a moment for the changes and send a message for the update. In the example above, it shows sth like added car recommendation module so we can see the updated commit on the github. For git push, it uploads everything to the Github repo in order for reviewing.