How to run:

- Step 1: Download data. I am using 2 csv files and one of them, application_record.csv doesn't fit in Git Hub as it is over the 25 MB limit. Make sure that both files, application_record.csv and credit_record.csv are in the src before running. https://www.kaggle.com/datasets/rikdifos/credit-card-approval-prediction?select=credit_record.csv. The credit_record is already in there but verify this.
- Step 2: cargo run. Run cargo run in the terminal
- Step 3: when asked "Would you like to know the most influential attributes affecting credit risk? (Yes or No):"
    - Type "Yes"
    - Expected output: The most influential attributes are:
        - Annual Income = 0: 20.00%
        - Number of Children = No children: 13.82%
        - Marital Status = Married: 13.74%
        - Property Ownership = Y: 13.45%
        - Gender = F: 13.42%
- Step 4: when asked "Would you like to know an example of one of the attributes with the highest possible credit risk (there is a tie between several)? (Yes or No):"
    - Type "Yes"
    - Expected output: Attributes with the highest possible credit risk:
        - Gender: Male
        - Property Ownership: Yes
        - Number of Children: No children

        - Annual Income: <20,000
        - Marital Status: Married
        - Credit Risk Percentage: 27.57%
- Step 5: when asked “Would you like to calculate your own credit risk? (Yes or No):”
    - Type “Yes”
- Step 6: answer several questions when prompted. Follow specifications if have any or else it won’t process.
    - Ex.
        - Biological gender (Male or Female): Male
        - Property Ownership (Yes or No): No
        - Number of Children: 2
        - Annual Income (no commas): 60000
        - Marital Status (Single/Married/Other): Married
        - Your predicted credit risk percentage is 19.64%.
- Step 7: cargo test to run tests
    - Expected output:
        - running 4 tests
        - test tests::generateriskpercenttest ... ok
        - test tests::test_bfs ... ok
        - test tests::testcalculateweights ... ok
        - test tests::testcreatecreditgraph ... ok
        -

    - test result: ok. 4 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
-

Code explanation:

Overview: As someone who has been denied credit, I have recently found extreme interest in credit and factors that make one more or less eligible for credit. To do this, I used a credit dataset from Kaggle (url: https://www.kaggle.com/datasets/rikdifos/credit-card-approval-prediction?select=credit_record.csv). I am interested in what features of people affect this and I used this csv to do this. This has two csv files, application_record.csv which has 439,000 rows and 10 columns in the compact. However, I am most interested in the gender, property ownership, number of children, income, and marital status so these were the columns I pulled from this dataset. The credit_record.csv has only 3 columns, the ID, the month's balance, and the credit status. This has 4 modules, a main.rs, an extremefeautres.rs, and inputsandload.rs, and a graphs.rs. I imported the csv dependency in my cargo.toml as well for reading csv’s. I also included a gitignore so that my csv file that is two big doesn’t get transferred to github. Essentially, this answers 3 questions, what are the attributes leading to the most risk, what is an example of a person with the highest risk, and what does your risk look like. All of these questions are answered in the scope of this code and are confined to the features I selected as I believe that these are the most interesting and universal amongst people. Ultimately, my data shows results for my dataset and gives a general idea of what features increase credit risk the most. Nevertheless, my code answers these questions while adding an interactive component that allows users to see the estimate for themselves.

main.rs:

To start I declared 3 separate modules, extremefeatures, inputsandload, and graphs that have different functions. Then I imported specific functions from each, such as the findextremerisk and generateriskpercent from extremefeatures, load,getinput, numberinput, and children categories from inputs and load, the createcreditgraph and calculateweights from graphs, along with std::error::Error for error handling. In my main function, I decided to return a Result where if it runs, it will end with Ok(()) and if it doesn't work, it handles errors effectively using a boxed dynamic error. I did this because all of my outputs are print statements so if it prints all of the print statements, Ok(()) will be run. To start, I load the data using the load function from my inputsandload module. I added a ? after to handle errors if for some reason the csv doesnt load. Then, I create a graph from the creditrecord csv and again used ? to handle errors more efficiently. Then, I called the bfs method on the graph which does a breadth first search to identify clients with high credit risk based on the graph. This is what sees which clients are at risk. Then, I use my calculate weights function to calculate the significance of different outcomes and factor that in. Then, I make the weights into a hashmap and sore the weights in descending order. This sorts and stores the weights. Then, I ask my first user question if they would like to know the most influential attributes. If they say yes, it prints the top 5 using .take(5). Then, it asks if the user would like to know an example of the highest risk. Because I only used 5 features, some are tied at 27.57% risk so this is an example of one of the people types with this highest risk. It does this by using my findextremerisk and generateriskpercent which find highest attribute combinations and then calculates the risk respectively. Then, I make the custom risk prediction, which asks questions such as gender, property ownership, number of children, annual income, and marital status and then uses generateriskpercentage to calculate the risk. Then, I created 4 tests in this module to test this. My first test, generateriskpercenttest, tests if the generateriskpercent calculates risk correctly. To do this, I make a mock feature and its weights and see if the generateriskpercent function calculates the total risk correctly. Then, my testbfs tests the bfs and correctly identifies at risk clients. It has one entry with 1 which means one month of missed payment and one with C meaning that all are paid. It then checks that risky has 1 and that the one marked C is not listed as risky. My third test, testcalculateweights, checks that calculate weights correctly calculates weight. It does this by creating a vector with two clients which different attributes. Then, it puts 1 at risk and then calls the calculate weights to compute weights. Then, it checks that the weights has the feature Male in the weights as this was a feature in the the risky person. Lastly, my last test checks that the createcreditgraph function works. I created a csv file using fs::write which was a new concept to me. I learned this in the rust dictionary. Then, I created a graph from this text file using createcreditgraph and checked that the adjacency list has a node for both clients. Then, also from outside research in the rust dictionary, I delete the file using fs::remove_file so the file isn't kept and I dont have to delete it every time manually. Throughout this assignment I tried to go above and beyond with my error handling and I did this through making my functions a result with one output being the correct output and if it doesnt work, returning a boxed error. I used this example: https://doc.rust-lang.org/rust-by-example/error/multiple_error_types/boxing_errors.html.

graphs.rs:

In this module, I use HashMapsm Hashsets, Error handling, and fs for files. I start by creating a struct called Graph with one field inside of it. This fiend is an adjacency list created by a hashmap with a string representing the ID and the vector of string representing the status. Then, I define an implementation that has a constructor called new and creates a new instance of the adjacency list. Then, I add edges using my addedge function. This looks for id in adjlist and uses .or_insert_with to crete a new vector for that key if it is not already created. If it is in there it adds the status to the client's vector. Then my bfs function performs abreadth first search. The output is a Hashset of risky clients. To do this it iterates through the adjacency list and loops through each client and their status. Then it checks if the status is anything but C or X (which show that the client isn't in credit risk) and deem it risky if it is anything but C or X. Then, my createcreditgraph, which returns a graph, or a boxed error if it does not work. I was initially troubled with this as I had no error handling but I fixed this as I went on and made the output a result to fix this. Then, I read the file and skip the header using .skip. I then split the lines and make sure that it is in the correct format by checking that the fields are larget than 3. Then I add edges and return the graph. Lastly in this module, I calculate the weights, which take in clients which is a struct of the persons data, and the clients at risk to show all clients at risk. Then, my raw function is initialized to count occurrences of each pair of risky clients and the numatrisk tracks this. Then, I go though the clients and create a vector for each and add to the raw for each. Then, I calculate the weight for every value in raw, I take the count over the number at risk to calculate this specific weight. I make it a percentage by multiplying itby 100.

<u>extremefeatures.rs:</u>

This module has two functions, findextremerisk and generateriskpercent. The findextremerisk function finds the highest credit risk by finding the combination of features that crete the highest risk percentage. Although there is a tie, it takes the first one. It does this through nested loops that iterate over all possible combinations of risk and then for each, creates a features vector and calls generateriskpercent on it. If it is bigger than what already is the extreme, this becomes the new extreme. Then, my generateriskpercent function, calculates the risk for a given combination based on weights. It takes in the features and the weights as inputs and calculates the risk. It first initializes the riskscore and then iterates over features while checking and getting weights using .get. In then adds it to risk score. Another safety measure that I didn't need was capping it at 100 so someones risk couldnt go over 100 using .min(100).

inputsandload.rs:

My last module inputsandload, loads client records, is code that allows the program to take in inputs for both strings and numbers, and categorizes the number of children for simplicity. To start, I created a Clientrecord struct that represents a client record with the fields that I selected as most interesting. All are strings but the income. I also used debug for formatting and partialeq for comparison. My first function is my load function that reads a csv file and creates a vector of Clientrecord out of it. To do this it takes in a path and returns a result, with the Clientrecord if it works and a box error if it doesn't and the file isn't able to be read. I used fs to read the file to string and use split to parse the values. I also added a safety measure to check that the len was greater or equal to 9 so that I knew the right info was going in. Then, it returns the records vector which is a vector of the clientrecord instances. Then, my getinput function is what allows for the user to add text input and returns as a string. It takes in a prompt and outputs the user response. It prints the prompt and uses trim to take out whitespace. Then, my number input function does a very similar thing but for an unsigned 32 bit integer. Lastly, my children categories takes in the number of children and outputs a string of the range of children. For this, I used a match function to match input with the number of children. In this, I made 0, 1 and 2, 3 and 4, and 5 and up the bins as this captured effectively. Nevertheless, these help with loading data, taking in u32 and string user input, and categorizing the number of children.

Nevertheless, by answering these questions, my code can give users an insight into credit insights on who is risky and what are attributes of risky people. It is obvious that owning a property dramatically increases this and more income definitely helps. Furthermore, being married significantly increases risk, especially for men, which is important to note. There are so many new and interesting insights from this and I hope you enjoy using it.