A. Project Overview

- **Goal**: What question are you answering?

  The question I am answering through this project is looking at how the death rate changes throughout the years. I'm looking at the death rate for each year and each country, both genders, and for all ages under age group (since it was divided into multiple age groups).

- **Dataset**: Source, size (link if too large for GitHub).

  I got my dataset from tableau where they have free public datasets for analysis. The specific data I used for my project is: a public health data set.

B. Data Processing

- How you **loaded** it into Rust.
- Any **cleaning** or **transformations** applied.

I first downloaded the dataset from the website (downloading the linked file called "Global Burden of Disease" under Free Health Data Sets). I then added this dataset into my "proj" directory inside of my ds210project folder so that my Rust functions would be able to access it. I renamed the csv to a shorter and simpler name called "health.csv". I didn't clean the data prior to loading it, I cleaned/filtered the data through my Rust functions.

C. Code Structure

1. **Modules**
   - Purpose of each and rationale for organization.

   My Rust project is organized into 4 modules: main.rs, dataframe.rs, linear.rs, and plot.rs which handles different things for my project.

   - Dataframe.rs = creates a Dataframe structure from a CSV, handling the reading of the CSV, loading, storing, and also operations like printing and filtering the data.
   - Linear.rs = handles training a linear regression model given data taken from the Dataframe given the column names through the linfa and linfa_linear crates. After locating the position of those columns in the dataset, it extracts it and converts it to ndarrays to fit the model.
   - Plot.rs = generates a scatter plot given the two selected columns and also puts a fitted regression line through it using the plotters crate to visualize it. The plot gets saved as an image file and the linear regression equation based on this model also gets printed.

- Main.rs = it runs the whole thing: loading the data from the csv I uploaded, filtering it (age group and gender), training a linear regression model based on the data, and generating a plot of it. It also has tests to verify the code is functioning the way it should.

The project is organized into these different modules so that each can handle a different task to make it more manageable, to fix, and to read.

2. **Key Functions & Types (Structs, Enums, Traits, etc)**
    - For each non-trivial item, restate:
        - **Purpose**
        - **Inputs/Outputs**
        - **Core logic and key components**

For my dataframe.rs:

I have a Dataframe struct which will store the labels, data, and the column types. The ColumnVal enum supports the different types of data in the columns, and for this data it represents one of 3 types: string, integer, or a float.

My read_csv function will load the csv into a Dataframe with the inputs of a file path and a list of expected column types. It reads the labels and rows, parses the values based on type, and then stores it as ColumnVal values. It returns OK(()) if it's successful or an error if not.

My filtering function will return a new Dataframe with just the rows that match the conditions. It takes the column label and a filtering closure as the input and outputs a filtered Dataframe. It gets the index of the specified column, applies the closure to that column, and returns the matching rows.

For my Linear.rs:

My fit_model function will fit a linear regression model to the two columns I specified. It takes the data, labels, x column name, and y column name as inputs. It will output a trained FittedLinearRegression<f64> model by taking the values from the selected x and y column, creating ndarray dataset, and fitting it through linfa linear regression.

For my plot.rs:

My plot_data function will create a scatter plot with a regression line, and print the regression line equation. It takes the data rows, labels, x axis, and y axis column names as inputs. The output is a generated png and returns ok(()) on success. It finds the right columns and filters the data, and uses plotters to do the scatter plot and fit the line. It then calls fit_model to get the slope and intercept for the equation.

3. **Main Workflow**
    ○ At a glance, how modules/functions interact to produce your results.

For my main.rs:

It loads the dataset through calling read_csv function and applies the filter function twice by age and gender (to get the desired values for those columns). It calls plot_data to visualize the data & regression line, and also print the equation for it. Main uses the dataframe.rs module to process the data, uses the linear.rs module to analyze it, and the plot.rs to visualize it.

D. Tests

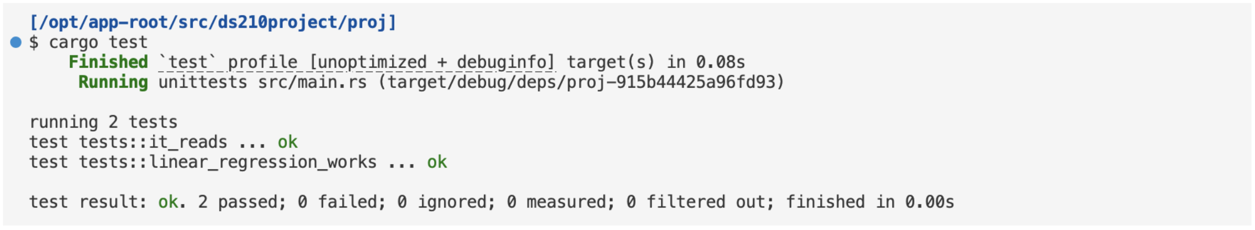
- `cargo test` **output** (paste logs or provide screenshots).
- For each test: what it checks and why it matters.

I have two tests for this project: one that checks if it reads the csv correctly and one that checks whether the linear regression outputs the expected slope and y-intercept.

For the csv test, it should return ok(()) meaning there is no error and the labels in the Dataframe match the one in the csv. It matters because it sets a foundation for the other functions. If it doesn't read or label the csv correctly, it could affect the filtering, plotting, and modeling linear regression.

For the linear regression test, it tests if the linear regression model accurately fits the data. To test this one, I made a sample csv called "test_data.csv" where the x and y values are computed based on the equation y = 5x + 50. I call the fit_model function to train the columns with those x and y values to check if it learns that the slope is approximately 5.0 and y intercept is approximately 50.0. (It is approximate because the floats can have small rounding errors).

My cargo test output:

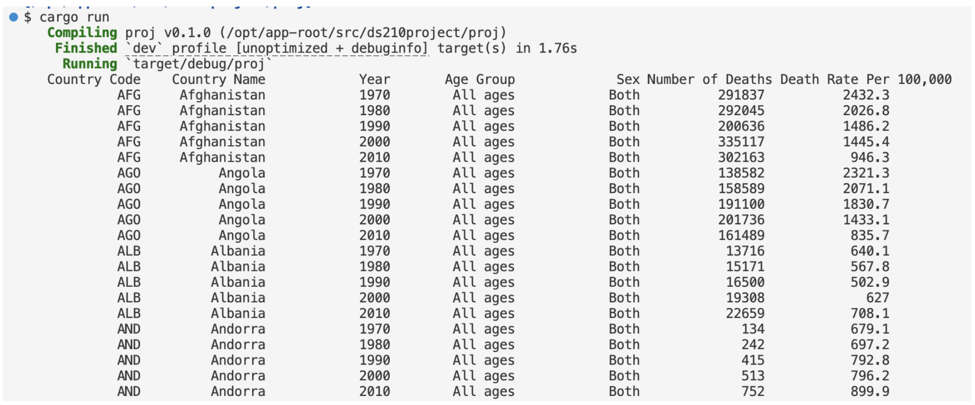


```
[/opt/app-root/src/ds210project/proj]
$ cargo test
    Finished `test` profile [unoptimized + debuginfo] target(s) in 0.08s
     Running unittests src/main.rs (target/debug/deps/proj-915b44425a96fd93)

running 2 tests
test tests::it_reads ... ok
test tests::linear_regression_works ... ok

test result: ok. 2 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

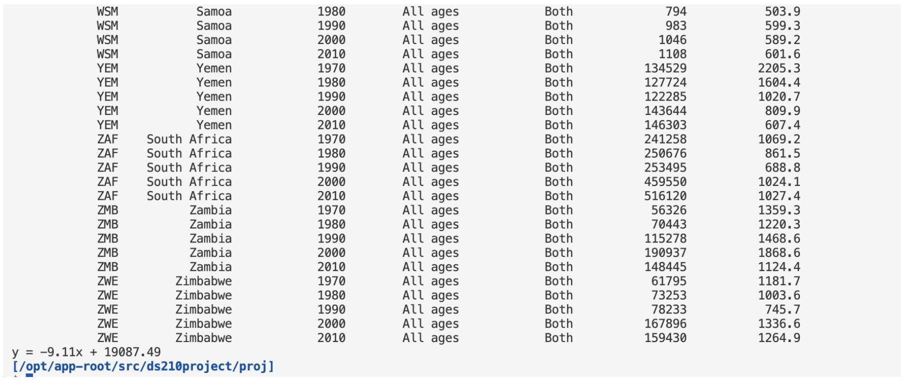
E. Results

- All program outputs (screenshots or pasted).
- Interpretation in project context (no need for "groundbreaking" results).

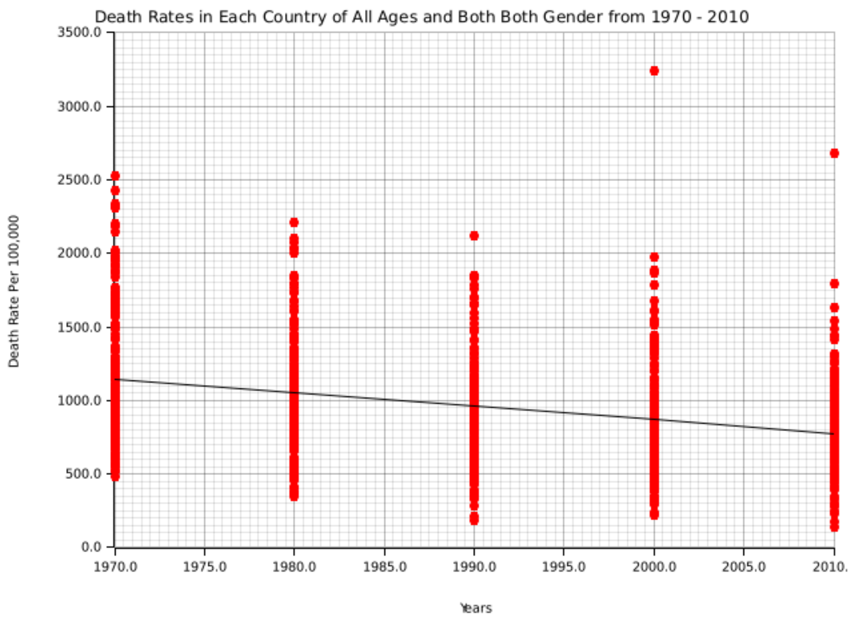
My output for the filtered dataset & plotting the results:



```
$ cargo run
   Compiling proj v0.1.0 (/opt/app-root/src/ds210project/proj)
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 1.76s
     Running `target/debug/proj`
   Country Code    Country Name           Year       Age Group            Sex Number of Deaths Death Rate Per 100,000
            AFG     Afghanistan           1970        All ages           Both         291837          2432.3
            AFG     Afghanistan           1980        All ages           Both         292045          2026.8
            AFG     Afghanistan           1990        All ages           Both         200636          1486.2
            AFG     Afghanistan           2000        All ages           Both         335117          1445.4
            AFG     Afghanistan           2010        All ages           Both         302163           946.3
            AGO          Angola           1970        All ages           Both         138582          2321.3
            AGO          Angola           1980        All ages           Both         158589          2071.1
            AGO          Angola           1990        All ages           Both         191100          1830.7
            AGO          Angola           2000        All ages           Both         201736          1433.1
            AGO          Angola           2010        All ages           Both         161489           835.7
            ALB         Albania           1970        All ages           Both          13716           640.1
            ALB         Albania           1980        All ages           Both          15171           567.8
            ALB         Albania           1990        All ages           Both          16500           502.9
            ALB         Albania           2000        All ages           Both          19308             627
            ALB         Albania           2010        All ages           Both          22659           708.1
            AND         Andorra           1970        All ages           Both            134           679.1
            AND         Andorra           1980        All ages           Both            242           697.2
            AND         Andorra           1990        All ages           Both            415           792.8
            AND         Andorra           2000        All ages           Both            513           796.2
            AND         Andorra           2010        All ages           Both            752           899.9
```



```
           WSM            Samoa          1980        All ages           Both             794          503.9
           WSM            Samoa          1990        All ages           Both             983          599.3
           WSM            Samoa          2000        All ages           Both            1046          589.2
           WSM            Samoa          2010        All ages           Both            1108          601.6
           YEM            Yemen          1970        All ages           Both          134529         2205.3
           YEM            Yemen          1980        All ages           Both          127724         1604.4
           YEM            Yemen          1990        All ages           Both          122285         1020.7
           YEM            Yemen          2000        All ages           Both          143644          809.9
           YEM            Yemen          2010        All ages           Both          146303          607.4
           ZAF     South Africa          1970        All ages           Both          241258         1069.2
           ZAF     South Africa          1980        All ages           Both          250676          861.5
           ZAF     South Africa          1990        All ages           Both          253495          688.8
           ZAF     South Africa          2000        All ages           Both          459550         1024.1
           ZAF     South Africa          2010        All ages           Both          516120         1027.4
           ZMB           Zambia          1970        All ages           Both           56326         1359.3
           ZMB           Zambia          1980        All ages           Both           70443         1220.3
           ZMB           Zambia          1990        All ages           Both          115278         1468.6
           ZMB           Zambia          2000        All ages           Both          190937         1868.6
           ZMB           Zambia          2010        All ages           Both          148445         1124.4
           ZWE         Zimbabwe          1970        All ages           Both           61795         1181.7
           ZWE         Zimbabwe          1980        All ages           Both           73253         1003.6
           ZWE         Zimbabwe          1990        All ages           Both           78233          745.7
           ZWE         Zimbabwe          2000        All ages           Both          167896         1336.6
           ZWE         Zimbabwe          2010        All ages           Both          159430         1264.9
y = -9.11x + 19087.49
[/opt/app-root/src/ds210project/proj]
```



- The filtered dataset only has data for all ages from the age group column and both gender from 1970 - 2010 for each of the countries. (there is more printed but in the screenshot I have the beginning and last few)
- The plot_data produces a visualization of the results and also the regression line equation. The slope is negative meaning that the death rate has decreased over the

years. The line in the plot is fitted quite well (going through where the average dots seems to be at) and also shows this decrease in death rates with the downward trend.

F. Usage Instructions

- How to build and run your code.
- Description of any command-line arguments or user interaction in the terminal.
- Include expected runtime especially if your project takes a long time to run.

You will just need to do cargo run to run this program. It runs quickly, probably taking a few seconds.

G. AI-Assistance Disclosure and Other Citations

- Cite any **substantive** ChatGPT/GenAI you used (e.g. screenshot or description).
  - You can skip this for common knowledge/debugging use cases
- For each cited snippet, include your own explanation to show understanding.
- You can also provide links to other sources you found useful that are not "common knowledge"

For plotting:

https://docs.rs/plotters/latest/plotters/#plotting-in-rust
https://chatgpt.com/share/680ee033-e198-8001-b30a-569b0e6ecb0b

I've used chatgpt to understand how the code to plot a scatter plot works (I took the code from the first link to plot and asked chatgpt to explain it) to which I then adjusted the code to fit my data. I also used it to fix the errors I was having when running the plot_data function. I understood now the different components of plotting the data and how to generate an image file such as .build_cartesian_2d() to define the x and y ranges for the chart, .draw_series() to determine the type you want (there are LineSeries, Circles, etc), .caption() to set a title, and others. To extract my x and y values, I could use match statements to match the enum so I can get the values. I knew that I needed to locate the index of my columns, but wasn't sure if there was a method to do so. Now, I know that I could iterate through my string of labels and through .position() find the position of that column label. When I was running into an error with the way I was passing my points that needed to be plotted into the Circle::new() method, I also used chatgpt for help. I was getting that error because points was a Vec<(f64,f64)> but Circle::new() expects an iterator of points and to turn a vec into an iterator, I could use .into_iter() and then map each of the x and y tuple to a Circle element.

For test functions:

https://chatgpt.com/share/6812e74d-c408-8001-8fc9-db8e595d869a

I also used chatgpt for understanding the parameters for my assert!() macro to check if my condition is true. I wasn't sure what would go inside the assert!() for a function that returns ok(()). I found out that for those I could use the is_ok() method.

https://chatgpt.com/share/681449d2-26dc-8001-8d37-9195e4a93310

I ran into an error with floating-point precision and not being able to represent the numbers exactly. In my code, I was using asssert_eq!() to do the comparison, but it would not work even if it was very close, it had to be exact for it to run successfully. Instead of checking if it's exact, I could check if it's approximately equal to 50.0. I would retrieve the y intercept and slope from the fit_model function and get the difference from the expected value, use .abs() to get the absolute value of that difference, and see if that value is very small.

I've used code from lecture 35 for plotting the points and the linear regression line. I also used similar code to convert my csv into a dataframe from homework 8. I have also received help at office hours from TAs, CAs, and professors.