Collaborator: Seo-young Yang
For this project, I wanted to find the correlation between the imdb ratings and meta scores of the movies. To do so, I chose to use the linear regression and find the R-squared value along with the polynomial regression for degree 2 and calculate the predictions.

This code should be run with the csv file in the same github file (imdb_top_1000.csv).

Cargo.toml should have the following dependencies:

```
[dependencies]
csv = "1.2"
nalgebra = "0.31"
plotters = "0.3"
```

main.rs

in the main, i wrote by calling functions from other modules with:

```
mod data;
mod models;
mod visualization;
```

so I can have the functions split.

Then I used this code:

```
use data::load_dataset;
use models::{fit_linear, predict_linear, predict_polynomial, calculate_r_squared};
use visualization::plot_predictions;
```

to import specific functions from the modules for use in main.rs.

With these lines:

```
fn main() -> Result<(), Box<dyn std::error::Error>> {
    let (imdb_ratings, meta_scores) = load_dataset("imdb_top_1000.csv")?;
    println!("Loaded dataset with {} rows.", imdb_ratings.len());
```

I was able to call the data set and read it from the CSV file and split data into imdb_ratings and meta_scores which are the independent variable and dependent variable that I chose to analyze. This line will calculate the slope and intercept of the linear regression line using fit_linear and display it.

The lines:

```
    let linear_predictions = predict_linear(&imdb_ratings, slope, intercept);
    let linear_r2 = calculate_r_squared(&meta_scores, &linear_predictions);
    println!("Linear Regression R²: {:.4}", linear_r2);
```

will predict meta scores using the linear model and calculate and display the R-squared value for the linear regression.

After this, I wrote:

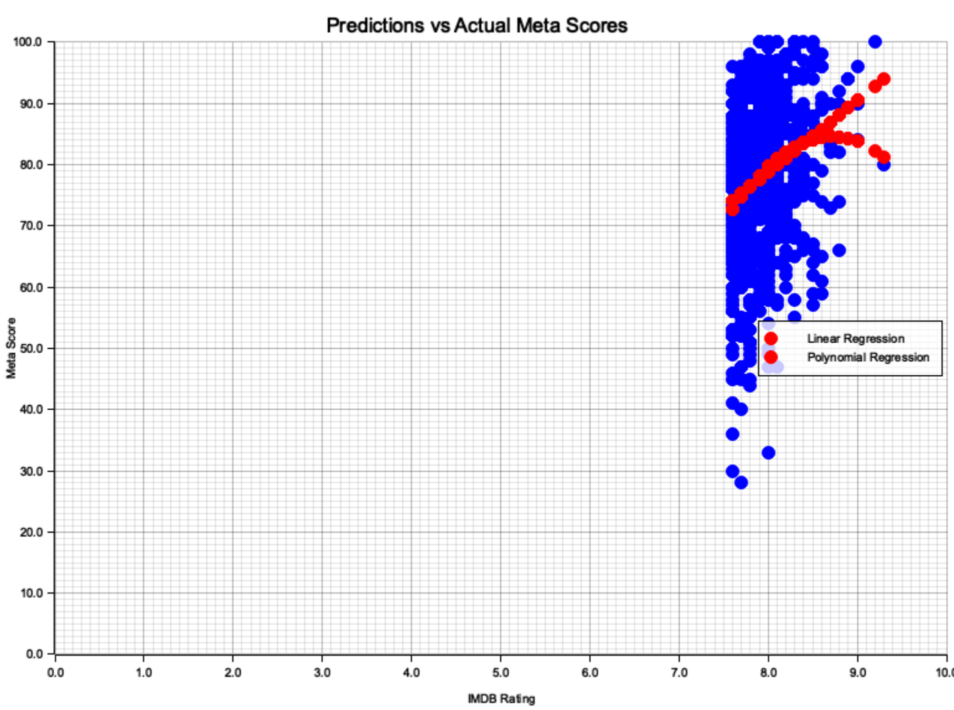
```
let polynomial_predictions = predict_polynomial(&imdb_ratings, &meta_scores, 2);
let polynomial_r2 = calculate_r_squared(&meta_scores, &polynomial_predictions);
println!("Polynomial Regression (Degree 2) R²: {:.4}", polynomial_r2);
```

to perform the polynomial regression for degree 2 and calculate the predictions.

Lastly,

```
plot_predictions(
    &imdb_ratings,
    &meta_scores,
    &[linear_predictions, polynomial_predictions],
    &["Linear Regression", "Polynomial Regression"],
    "predictions.png",
)?;
println!("Plot saved as predictions.png");
```

This will generate a plot that will allow me to compare the actual data with predictions using the plot_predictions and saved as predictions.png, which is this following picture.



Based on the returned values, which are:
Linear Regression Equation: y = 11.7130x + -14.9324
Linear Regression R^2: 0.0721
Polynomial Regression (Degree 2) R^2: 0.0811
Plot saved as predictions.png

We can say that with every 1-point increase in IMDB rating, the meta score increases by approximately 11.7130 points. Though negative y-intercept value is impossible, this model will predict this due to extrapolation. The R-squared value tells that the values are positively correlated, meaning that when IMDB rating increases, meta score will also increase, but the correlation is very low as only 7.21 % of the meta score can be explained by this correlation. The polynomial regression (degree 2) explains 8.11% of the variance in meta scores, which is only a slight improvement from the linear regression, which suggests that adding a quadratic term does not significantly improve the fit. This weak correlation may suggest that there might be additional factors, such as when it was released and audience preferences may affect the scores, and they might have inherently different criteria for evaluation.

data.rs

```
use csv::Reader;
use std::error::Error;
```

This will import the csv crate to read CSV files, including the error trait to handle the potential errors when reading the file.

```
pub fn load_dataset(file_path: &str) -> Result<(Vec<f64>, Vec<f64>), Box<dyn Error>> {
    let mut rdr = Reader::from_path(file_path)?;
    let (mut imdb_ratings, mut meta_scores) = (Vec::new(), Vec::new());
```

This will open the file at file_path and initialize empty vectors to store IMDB ratings and Meta scores from csv.

```
    for result in rdr.records() {
        let record = result?;
        if let (Ok(imdb), Ok(meta)) = (
            record.get(6).unwrap_or("").parse::<f64>(),
            record.get(8).unwrap_or("").parse::<f64>(),
        ) {
            imdb_ratings.push(imdb);
            meta_scores.push(meta);
        }
    }
```

This iterates through each record in the CSV and extracts the values from columns 6 and 8 and parse the values into f64.

```
    if imdb_ratings.is_empty() || imdb_ratings.len() != meta_scores.len() {
        return Err("Invalid dataset".into());
    }
    Ok((imdb_ratings, meta_scores))
}
```

This will ensure that the dataset is valid with no empty parts and matched lengths. Lastly, it will return the vectors as a tuple.

```
#[cfg(test)]
mod tests {
    use super::*;
```

This will make a test, so it can be used with other csv files if wanted.

```
#[test]
fn test_load_dataset() {
    let result = load_dataset("test_data.csv");
    assert!(result.is_ok(), "Dataset failed to load");
    let (imdb_ratings, meta_scores) = result.unwrap();
    assert_eq!(imdb_ratings.len(), meta_scores.len(), "Dataset lengths do not match");
    assert!(imdb_ratings.len() > 0, "Dataset is empty");
}
```

This will unwrap the result to get the imdb_ratings and meta_scores vectors. So it will ensure that the dataset loads successfully and the vectors for IMDB ratings and meta scores are properly aligned and non-empty.

```
#[test]
fn test_invalid_dataset() {
    let result = load_dataset("non_existent.csv");
    assert!(result.is_err(), "Invalid dataset should return an error");
}
```

This will handle missing or invalid files and errors should be returned.

model.rs

```
pub fn calculate_r_squared(actual: &[f64], predicted: &[f64]) -> f64 {
    let mean_actual = actual.iter().sum::<f64>() / actual.len() as f64;
    let ss_total = actual.iter().map(|&y| (y - mean_actual).powi(2)).sum::<f64>();
    let ss_residual = actual
        .iter()
        .zip(predicted.iter())
        .map(|(&y, &y_pred)| (y - y_pred).powi(2))
```

```
        .sum::<f64>();
    1.0 - (ss_residual / ss_total)
}
```

This will measure how much variance in actual is explained by the predictions by calculating the R-squared value.

```
pub fn fit_linear(imdb_ratings: &[f64], meta_scores: &[f64]) -> (f64, f64) {
    let (mean_x, mean_y) = (
        imdb_ratings.iter().sum::<f64>() / imdb_ratings.len() as f64,
        meta_scores.iter().sum::<f64>() / meta_scores.len() as f64,
    );
```

This will calculate the mean of imdb_ratings and meta scores.

```
    let slope = imdb_ratings
        .iter()
        .zip(meta_scores.iter())
        .map(|(&x, &y)| (x - mean_x) * (y - mean_y))
        .sum::<f64>()
        / imdb_ratings.iter().map(|&x| (x - mean_x).powi(2)).sum::<f64>();
    (slope, mean_y - slope * mean_x)
}
```

This will compute the slope and intercept.

```
pub fn predict_linear(imdb_ratings: &[f64], slope: f64, intercept: f64) -> Vec<f64> {
    imdb_ratings.iter().map(|&x| slope * x + intercept).collect()
}
```

This applies the equation to each IMDB rating and collects the results into a vector.

From here, it will predict the polynomial

```
pub fn predict_polynomial(imdb_ratings: &[f64], meta_scores: &[f64], degree: usize) -> Vec<f64> {
    let x_matrix: Vec<Vec<f64>> = imdb_ratings
        .iter()
        .map(|&x| (0..=degree).map(|i| x.powi(i as i32)).collect())
        .collect();
```

This constructs a matrix where each row contains powers of an IMDB rating.

```
    let x_matrix = DMatrix::from_row_slice(imdb_ratings.len(), degree + 1, &x_matrix.concat());
    let y_vector = DVector::from_column_slice(meta_scores);
    let x_t = x_matrix.transpose();
```

```
    let coefficients = (x_t.clone() * &x_matrix)
        .try_inverse()
        .unwrap()
        * x_t
        * y_vector;
    (x_matrix * coefficients).as_slice().to_vec()
}
```

This uses the least squares method to calculate the polynomial coefficients and predicts values by multiplying the coefficients with the matrix.

```
#[test]
fn test_fit_linear() {
    let imdb_ratings = vec![1.0, 2.0, 3.0, 4.0, 5.0];
    let meta_scores = vec![2.0, 4.0, 6.0, 8.0, 10.0];
    let (slope, intercept) = fit_linear(&imdb_ratings, &meta_scores);
    assert!((slope - 2.0).abs() < 1e-6, "Incorrect slope");
    assert!((intercept - 0.0).abs() < 1e-6, "Incorrect intercept");
}
```

This confirms the fit_linear function calculates correct parameters for a perfectly linear dataset.

```
#[test]
fn test_predict_linear() {
    let imdb_ratings = vec![1.0, 2.0, 3.0];
    let slope = 2.0;
    let intercept = 1.0;
    let predictions = predict_linear(&imdb_ratings, slope, intercept);
    assert_eq!(predictions, vec![3.0, 5.0, 7.0], "Linear predictions are incorrect");
}
```

Ensures predict_linear correctly applies the regression equation.

```
#[test]
fn test_predict_polynomial() {
    let imdb_ratings = vec![1.0, 2.0, 3.0];
    let meta_scores = vec![1.0, 4.0, 9.0];
    let predictions = predict_polynomial(&imdb_ratings, &meta_scores, 2);
    assert!((predictions[0] - 1.0).abs() < 1e-6, "Polynomial prediction is incorrect");
}
```

Ensures predict_polynomial accurately fits and predicts for a quadratic dataset.

```
#[test]
fn test_calculate_r_squared() {
    let actual = vec![2.0, 4.0, 6.0];
```

```
    let predicted = vec![2.0, 4.0, 6.0];
    let r2 = calculate_r_squared(&actual, &predicted);
    assert!((r2 - 1.0).abs() < 1e-6, "R² should be 1.0 for perfect predictions");
}
```

Ensures calculate_r_squared correctly computes R-squared value for perfect predictions.

visualization.rs

```
pub fn plot_predictions(
    imdb_ratings: &[f64],
    actual_meta_scores: &[f64],
    predictions: &[Vec<f64>],
    labels: &[&str],
    file_name: &str,
) -> Result<(), Box<dyn Error>> {
```

This accepts data, predictions, labels, and a filename to save the plot.

```
    let root = BitMapBackend::new(file_name, (800, 600)).into_drawing_area();
    root.fill(&WHITE)?;
```

This sets up the plot canvas.

```
    let mut chart = ChartBuilder::on(&root)
        .caption("Predictions vs Actual Meta Scores", ("sans-serif", 20))
        .margin(20)
        .x_label_area_size(40)
        .y_label_area_size(40)
        .build_cartesian_2d(0.0..10.0, 0.0..100.0)?;
```

This configures the chart layout.

```
    chart.draw_series(
        imdb_ratings
            .iter()
            .zip(actual_meta_scores.iter())
            .map(|(&x, &y)| Circle::new((x, y), 5, ShapeStyle::from(&BLUE).filled())),
    )?;
```

This will plot the actual data points in blue points.

```
    for (pred, label) in predictions.iter().zip(labels.iter()) {
        chart
            .draw_series(
```

```
            imdb_ratings
                .iter()
                .zip(pred.iter())
                .map(|(&x, &y)| Circle::new((x, y), 5, ShapeStyle::from(&RED).filled())),
        )?
        .label(*label);
    }
```

This will overlay the prediction points with labels in red.

```
#[cfg(test)]
mod tests {
    use super::*;
```

This is also a testing part.

```
let imdb_ratings = vec![1.0, 2.0, 3.0];
let meta_scores = vec![10.0, 20.0, 30.0];
```

This will define test input data and vector contains three data points each.

```
let predictions = vec![vec![10.0, 20.0, 30.0], vec![12.0, 22.0, 32.0]];
let labels = vec!["Model 1", "Model 2"];
```

They are the predictions and labels the models.

```
let result = plot_predictions(
    &imdb_ratings,
    &meta_scores,
    &predictions,
    &labels,
    "test_plot.png"
);
```

This calls the plot_predictions function with the test data.

```
assert!(result.is_ok(), "Plotting function should not panic");
```

This will ensure that the function executes successfully without returning an error.

Conclusion

This project demonstrates the application of regression techniques and data visualization to analyze movie rating datasets. While the models showed limited predictive accuracy, in the project, I tried to include how I would like to analyze the data and from this, process the dataset and visualize. These findings lay a strong foundation for future work, emphasizing the importance of incorporating diverse features and exploring advanced modeling techniques to capture complex relationships in data. I might have also thought there would be correlations between the imdb score and meta score, but there might not have been since imdb is a user rating and meta score is by the professional critics, and since they might have different standards, it might have led to finding weak correlations.