Data Analysis and Visualization in Rust

Amay B · Following
Published in CoderHack.com
4 min read · Sep 15, 2023

Rust is a modern programming language focused on safety, speed, and concurrency. It provides powerful features around data manipulation, analysis, and visualization. This article will provide an overview of how to get started with data science workflows in Rust.

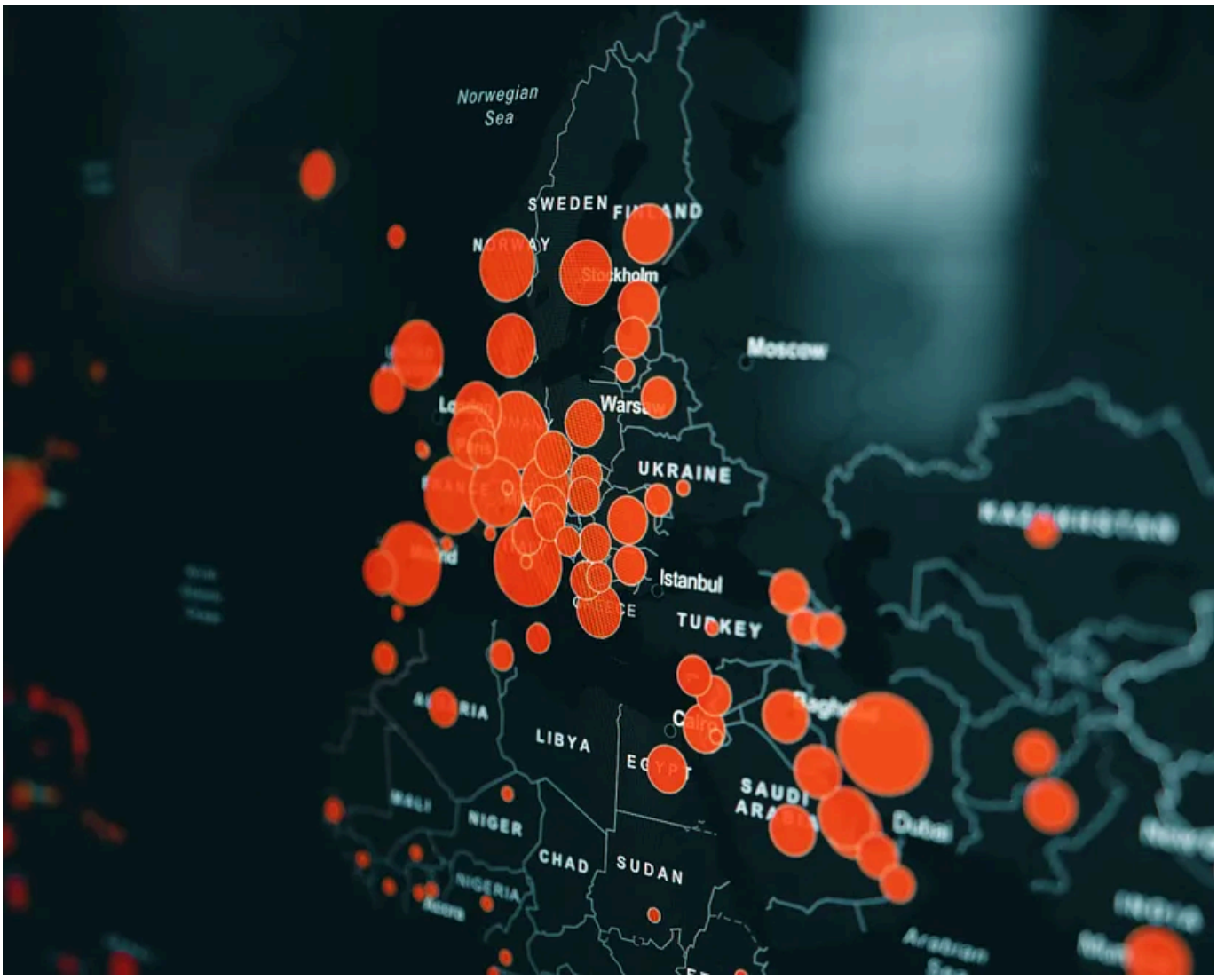

Photo by Clay Banks on Unsplash

Installation and Setup

To get started with Rust, install the Rust programming language and the Cargo package manager. Rustup is the preferred method to install Rust and Cargo.

Once Rust is installed, you'll need an IDE to write code in. Popular options for Rust include:

- Visual Studio Code: Free and open source, with good Rust support via extensions
- Atom: Also free and open source, with the rust-lang package for Rust support
- CLion: Cross-platform IDE from JetBrains with Rust support

Let's create our first Cargo project. Run:

```
cargo new data-science
```

This will create a new folder called `data-science` with some starter files.

Basic Data Types

Rust has a number of data types we can use for data analysis. Some key ones include:

Scalar Types

- Integers: u8, i32, usize, etc.
- Floating point numbers: f32, f64
- Booleans: true and false
- Characters: char

Compound Types

- Tuples: Fixed size collections of elements of different types.
- Arrays: Fixed size collections of elements of the same type.

Collections

- Vectors: Resizable arrays, like dynamic arrays.
- Hashmaps: Key-value data structure.

Here's an example of some basic types in code:

```
// Scalar types
let x = 5;     // i32
let y = 10.5;   // f64

// Compound types
let tup = (1, "Hello", 1.5); // Tuple
let arr = [1, 2, 3];        // Array

// Collections
let mut vec = Vec::new();   // Vector
vec.push(1);
vec.push(2);

let mut map = HashMap::new(); // Hashmap
map.insert("key", 1);
```

Reading and Writing Data

To read and write data, we'll use popular crates for data serialization. Some options include:

CSV

The csv crate can be used to read and write CSV data. For example:

```
extern crate csv;

fn main() {
  let mut rdr = csv::Reader::from_reader(io::stdin());
  for record in rdr.records() {
      // Use record
  }
}
```

JSON

The serde crate makes it easy to read and write JSON data. For example:

```
extern crate serde;
extern crate serde_json;

use serde_json::Value;

fn main() {
  let json_str = r#"{ "name": "John Doe" }"#;
  let v: Value = serde_json::from_str(json_str);

  println!("{}", v["name"]);
}
```

SQLite

The rusqlite crate provides a Rust wrapper around SQLite for reading and writing data to a database. For example:

```
extern crate rusqlite;

use rusqlite::Connection;

fn main() {
  let conn = Connection::open("test.db").unwrap();

  conn.execute(
    "CREATE TABLE person (
                name  TEXT,
                age INTEGER
              )",
    []);
}
```

Data Manipulation

Once we have data loaded, we'll want to manipulate and explore it. Some common operations include:

- Filtering: Keeping only rows/values that match some condition.
- Sorting: Sorting data by one or more columns.
- Grouping: Aggregating values by categories.

- Joins: Combining data from multiple sources based on a common column.
- Transforms: Applying functions (map, iter, etc) to transform data.

The ndarray crate provides a lot of this functionality and works well with Rust's type system.

Exploratory Data Analysis

We can analyze data in a few ways:

- Summary statistics: Mean, median, quantiles, standard deviation, etc.
- Frequency tables: Count frequencies and proportions of categories.
- Correlation analysis: Calculate correlations between numeric variables using covariance.
- Hypothesis testing: Perform statistical tests like t-tests, chi-squared tests, etc.

The statrs crate provides a lot of statistical functionality for data analysis in Rust.

Data Visualization

There are a few crates for visualization in Rust:

- plotters: Powerful 2D plotting library. Can create line charts, bar charts, histograms, box plots, heatmaps, etc.
- vega-lite: Bindings to the Vega-Lite visualization grammar for interactive charts.
- grisu-plot: Another option for 2D visualization with support for plotting from CSV/TSV data.

Here's a simple example of a line chart with plotters:

```
extern crate plotters;

use plotters::prelude::*;

fn main() {
  let mut data = Vec::new();
  data.push((1., 2.));
  data.push((2., 3.));
  data.push((3., 1.));

  let mut chart = ChartBuilder::on(&data)
                            .caption("Example plot", ("sans-serif", 50))
                            .set_label_area_size(LabelAreaPosition::Left, 40)
                            .line_chart();

  chart.configure_mesh()
     .draw()
     .unwrap();
}
```

Machine Learning

There are a few crates for machine learning in Rust:

- rustlearn: Machine learning library with classification (logistic regression, naive bayes, decision trees), regression (linear regression, lasso, ridge), clustering (K-means), etc.
- ndarray: N-dimensional arrays, useful for ML. Provides a lot of ML functionality too.

- nom: Parser for training data.
- linfa: Linear algebra library for ML.

Here's an example of logistic regression from rustlearn:

```
extern crate rustlearn;

use rustlearn::{Dataset, LogisticRegression, Metric};

fn main() {
  let mut dataset = Dataset::new();
  dataset.push([-1., -2.], -1);
  dataset.push([0., 1.], 1);
  dataset.push([2., 1.], 1);

  let mut lr = LogisticRegression::default();
  lr.fit(&dataset, Metric::Accuracy);

  let pred = lr.predict(&[-1., -2.]);
  assert_eq!(pred, -1);

  let pred = lr.predict(&[1., 2.]);
  assert_eq!(pred, 1);
}
```

Deploying Models

To use models in production, we need to be able to export and load them. Some options for this include:

- Export to a file (JSON, Protobuf, etc) and load the data back
- Export to a database and query the database to load the model
- Host the model as an API endpoint to get predictions
- Use a format like ONNX to export and load models

The rustlearn and rust-tensorflow crates provide some model exporting functionality.

Additional Topics

Some additional topics that could be covered include:

- Parallelism and concurrency in Rust for performance
- Building web APIs to serve data and models
- Using databases like PostgreSQL, MongoDB, etc for larger datasets
- Exploring other domain specific crates for finance, natural language processing, etc.

Rust has a lot of promise for data science with its combination of safety, performance and rich ecosystem of data-centric crates. I hope this overview provides a sense of how Rust can be used for a data analysis and visualization workflow!

Artifical Intelligence | Rust | Programming