GitHub Repository Language Analysis

Data Analysis Capstone Project

Executive Summary

This project analyzes programming language trends and patterns across GitHub repositories to understand how developers choose technologies and what factors influence language adoption. Using GitHub's API, so far i've collected 1200 repositories to answer questions about language popularity, project characteristics, and developer behavior.

Main Focus: Understanding what makes certain programming languages more popular than others and how language choice relates to project success, team size, and project type.

Data Source: GitHub REST API providing repository metadata, language statistics, and project characteristics across diverse open-source projects.

Deliverables: Interactive dashboard showing language trends, statistical analysis of factors influencing language choice, and practical insights for developers and hiring managers.

Value: Helps developers make informed technology choices, assists recruiters in understanding skill demand, and provides data-driven insights into software development trends.

Motivation

I chose GitHub language analysis because it combines my interest in programming with practical data analysis skills. The results will be useful for my own career decisions and interesting to other developers.

Why This Matters:

- **Career Planning**: Understanding which languages are growing vs. declining
- **Project Planning**: What languages work best for different types of projects
- **Industry Insights**: Real data about how the development world actually works
- **Skill Development**: Working with APIs, cleaning messy data, creating clear visualizations

Learning Goals:

- Get comfortable with GitHub API and rate limiting
- Practice data cleaning and exploratory data analysis
- Create compelling visualizations and dashboards
- Draw meaningful insights from real-world data
- Present technical findings to non-technical audiences

Data Question

Initial Exploration Direction:

Do repositories written in different programming languages show significant differences in health metrics (commit frequency, issue resolution time, contributor activity, and maintenance consistency), and which languages are associated with the healthiest project ecosystems?

Questions:

1. **Language Popularity Trends**: Which programming languages are most common, and how does popularity relate to project activity (stars, forks, recent commits)?

2. **Project Success by Language**: What languages tend to get more stars, contributors, or forks?

3. **Project Types and Language Choice**: How do language preferences differ between web development, data science, mobile apps, and system programming projects?

4. **Repository Characteristics**: What's the relationship between programming language and repository size, team size, project age, and update frequency?

5. **Multi-Language Projects**: How common are multi-language repositories, and what combinations of languages frequently appear together?

6. **Geographic and Temporal Patterns**: Are there differences in language preferences by region or changes in language adoption over time?

I'll pick **ONE** of these as my main focus based on what's most interesting in the initial data exploration.

Minimum Viable Product (MVP)

Primary Deliverable: Interactive dashboard analyzing GitHub language patterns with clear, actionable insights.

Core Analysis Components:

1. Language Popularity Analysis

- Distribution of languages across repositories
- Popularity vs. activity metrics (stars, forks, commits)
- Growth trends for different languages
- Clear visualizations showing market share and momentum

2. Project Success Metrics

- Statistical analysis of what makes repositories successful
- Comparison of success metrics across programming languages
- Identification of high-performing language/project-type combinations

3. Repository Characteristics Study

- Relationship between language choice and project size, team size, activity
- Analysis of multi-language projects and common technology stacks
- Patterns in repository organization and structure

4. Interactive Dashboard

- **Tool**: Sql, Python, Jupyter, power bi or tableau
- **Features**: Filter by language, project type, time period
- **Visualizations**: Charts, maps, trend lines, correlation plots
- **User Experience**: Easy for non-technical users to explore

To make the interactive dashboard more robust, consider these additional details:

4. Interactive Dashboard

- **Tool**: SQL (for data extraction and initial processing), Python (with libraries like Pandas for data manipulation, Matplotlib/Seaborn/Plotly for advanced visualizations), Jupyter Notebooks (for iterative development and sharing code), and Power BI or Tableau (for building the final interactive dashboard with a user-friendly interface).
- **Features**:
    - **Filter by language**: Allow users to select one or multiple programming languages to analyze.
    - **Project type**: Categorize projects (e.g., open source, academic, personal, company) and enable filtering.
    - **Time period**: Provide options to view data by year, quarter, month, or a custom date range.
    - **Repository size/stars**: Add filters for repository popularity or size metrics.
    - **Topic/Keywords**: Implement a search or filter by project topics/keywords if available in the dataset.
- **Visualizations**:
    - **Charts**: Bar charts (top languages, project types), pie charts (distribution of license types), line charts (trends over time for language usage), scatter plots (correlation between stars and forks).
    - **Maps**: If location data is available for contributors or organizations, visualize project distribution geographically.
    - **Trend lines**: Show growth or decline in specific language adoption, project creation, or contributor activity over time.
    - **Correlation plots**: Explore relationships between different metrics (e.g., number of contributors vs. project size, language popularity vs. issues closed).
    - **Heatmaps**: Visualize the activity matrix of contributions across different repositories or timeframes.
    - **Gantt charts**: (Optional, if project milestones are tracked) to show project progress.
- **User Experience**:
    - **Easy for non-technical users to explore**: Design an intuitive interface with clear labels, tooltips, and straightforward navigation.
    - **Drill-down capabilities**: Allow users to click on a data point (e.g., a specific language) to see more detailed information related to it.
    - **Export options**: Enable users to export filtered data or visualizations in various formats (CSV, PNG, PDF).
    - **Responsive design**: Ensure the dashboard is accessible and usable on different devices (desktop, tablet).
    - **Performance optimization**: Optimize data queries and dashboard loading times to ensure a smooth user experience, especially with large datasets.
    - **Customizable views**: Potentially allow users to save their preferred filter settings or dashboard layouts.

Analysis Outputs:

Statistical Findings:

- Clear summary statistics for each major programming language
- Correlation analysis between language choice and success metrics
- Confidence intervals and significance testing where appropriate

Business Insights:

- Practical recommendations for developers choosing technologies
- Market analysis useful for recruiters and hiring managers
- Trend predictions based on historical data

Technical Documentation:

- Data collection methodology and limitations
- Analysis process and assumptions
- Reproducible code for all visualizations and statistics

Success Criteria:

- **Comprehensive**: Covers 15+ major programming languages
- **Accurate**: Based on solid statistical analysis of 1,000+ repositories
- **Actionable**: Provides clear insights that people can actually use
- **Engaging**: Interactive visualizations that tell a compelling story

---

Sources

Primary Source: GitHub REST API

What I'll Collect:

- **Repository Metadata**: Name, description, creation date, last update, owner type
- **Activity Metrics**: Stars, forks, watchers, contributor count, commit frequency
- **Language Data**: Language breakdown by bytes of code, primary language
- **Project Characteristics**: Repository size, file count, license type, topics/tags

Sampling Strategy:

- **Popular Repositories**: Top 100 repos for each major language (Python, JavaScript, Java, C++, Go, TypeScript, C#, PHP, Ruby, Swift, Kotlin, Rust)
- **Diverse Sizes**: Mix of small personal projects and large organizational repositories
- **Active Projects**: Focus on repositories updated within the last 2 years
- **Geographic Diversity**: Include repositories from different regions/organizations

Data Collection Plan:

- **Volume**: 1,200-1,500 repositories total
- **Storage**: CSV files for analysis, JSON backup for detailed data
- **Quality Control**: Manual spot-checking and automated data validation

API Rate Management:

- 5,000 requests/hour limit requires spreading collection over multiple days
- Implement caching and error handling for robust data collection
- Priority on most important metrics if rate limits become constraining

---

Known Issues and Challenges

Data Collection Challenges:

1. GitHub API Rate Limits

- **Problem**: 5,000 requests/hour limits data collection speed
- **Solution**: Spread collection over 3-4 days, implement smart caching
- **Backup**: Use existing GitHub datasets if API access becomes problematic

2. Repository Selection Bias

- **Problem**: Popular repositories aren't representative of typical development
- **Solution**: Mix popular repos with random sampling, acknowledge limitations
- **Reality Check**: Focus analysis on trends within the data I can collect

3. Data Quality Issues

- **Problem**: Empty repos, forks, generated code, inactive projects
- **Solution**: Clear filtering criteria and data cleaning pipeline
- **Documentation**: Be transparent about what I included/excluded and why

Analysis Challenges:

1. Correlation vs. Causation

- **Problem**: Can't prove that language choice causes project success
- **Solution**: Focus on patterns and correlations, avoid claiming causation
- **Insight**: This is good practice for real-world data analysis

2. Confounding Variables

- **Problem**: Project type, team experience, timing all affect success
- **Solution**: Control for obvious factors where possible, acknowledge limitations
- **Learning**: Understanding what you can't control is part of data analysis

3. Statistical Significance

- **Problem**: Need enough data to make meaningful comparisons
- **Solution**: Focus on major languages with substantial sample sizes
- **Backup**: Qualitative analysis for interesting patterns in smaller samples

Technical Challenges:

1. Dashboard Development

- **Risk**: Interactive dashboards can be time-consuming
- **Solution**: Start with static visualizations, add interactivity if time permits
- **Priority**: Clear insights matter more than fancy features

2. Data Visualization Complexity

- **Challenge**: Making technical data accessible to general audience
- **Approach**: Simple, clear charts with good explanations

- **Test**: Run visualizations by non-technical friends for feedback

Project Management:

Timeline Risk: 6-8 weeks is tight for data collection, analysis, and presentation **Mitigation**: Start data collection immediately, have backup plans for scope reduction **Success Strategy**: Deliver solid analysis on a focused question rather than superficial coverage of everything

Realistic Expectations:

- I'm a beginner, so I'll focus on doing solid basic analysis well
- Better to have clear insights on fewer questions than weak analysis on many
- The goal is demonstrating data analysis skills and learning, not groundbreaking research