**CS673 Software Engineering** 

**Team X - Project Name**

**Project Proposal and Planning**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Xuansheng Xia | Team Leader & Design and Implementation Leader | *Xuansheng Xia* | 09/26/2024 |
| Xiaojuan Li | Requirement Leader | *Xiaojuan Li* | 09/26/2024 |
| Hengyi Song | Security Leader | *Hengyi Song* | 09/26/2024 |
| Ziliang Ren | QA Leader | *Ziliang Ren* | 09/26/2024 |
| Mingyuan Xu | Configuration Leader | *Mingyuan Xu* | 09/26/2024 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| **Iteration 0** | **Xuansheng Xia, Xiaojuan Li, Hengyi Song, Ziliang Ren, Mingyuan Xu** | 09/26/2024 |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

[Overview](#_g6igqliy7rm)

[Related Work](#_bf21eadgjj29)

[Proposed High level Requirements](#_rgyo4hi9stmq)

[Management Plan](#_ts358bsdtbcv)

[Objectives and Priorities](#_nxeeppkjxgn4)

[Risk Management (need to be updated constantly)](#_tk7yixobah8p)

[Timeline (need to be updated at the end of each iteration)](#_iksrndohvx29)

[Configuration Management Plan](#_j5uvivmxqcsp)

[Tools](#_dzly5b9kz982)

[Deployment Plan if applicable](#_sd8zu6r3jisd)

[Quality Assurance Plan](#_vra5ptwu59qx)

[Metrics](#_vwjduhc9wuah)

[Code Review Process](#_hx3eaiwb8v3m)

[Testing](#_l9xnpmd6hh0y)

[Defect Management](#_5amsh8h9f0c7)

[References](#_pd9euov6m4du)

[Glossary](#_ty3i2nqffhtc)

# Overview

This project focuses on building a web-based platform for analyzing and visualizing movie data from IMDB. The motivation for this project arises from the need to process and present large datasets in a way that makes it easy to identify trends and gather insights. The film industry, in particular, generates extensive data related to movie releases, reviews, ratings, and production details, and this project aims to provide a structured solution for visualizing this information effectively.

The purpose of the platform is to allow users to explore IMDB movie data through various interactive visualizations. It will serve users such as movie enthusiasts, data analysts, film critics, and researchers who are interested in trends related to movie releases, ratings, genres, and reviews. The system will feature core functionalities such as visualizing movie release trends by year, genre, and production country, as well as performing sentiment analysis on user reviews and generating word clouds based on common terms found in the reviews.

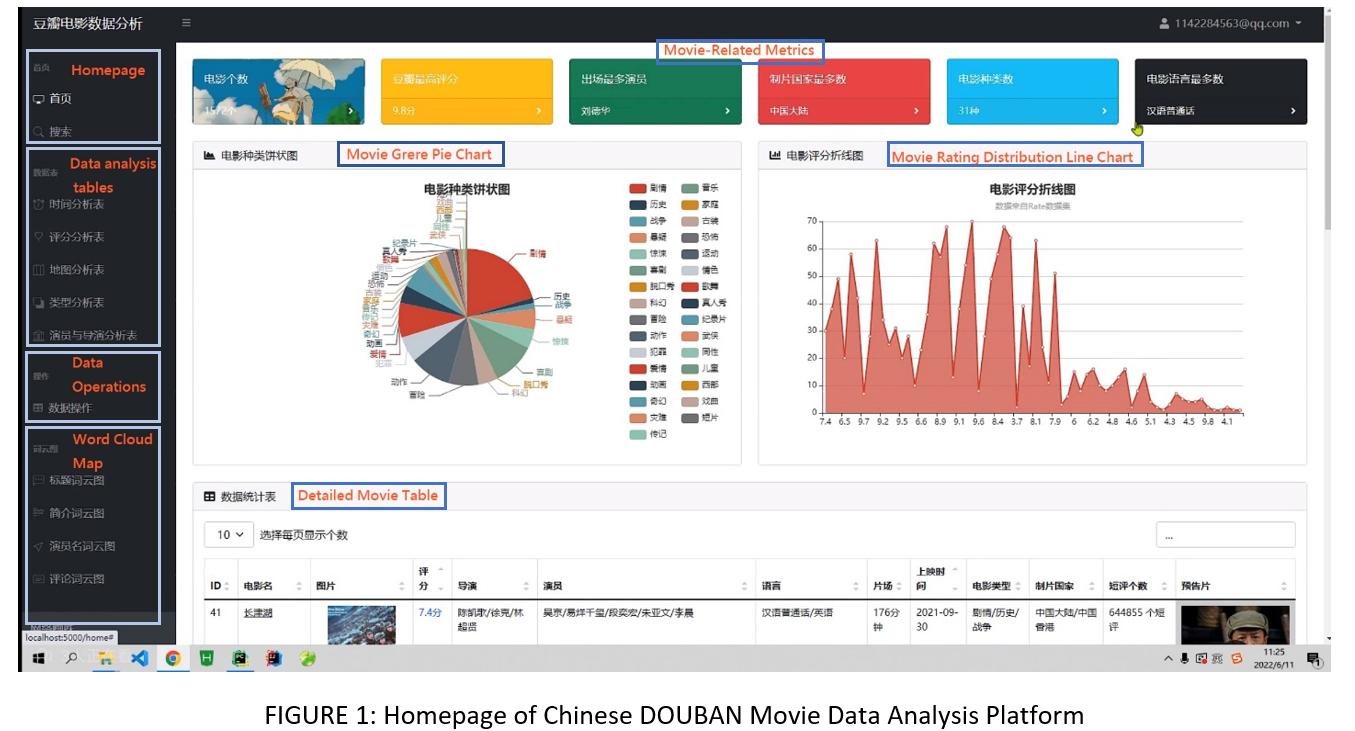
To implement these features, the project will use a technology stack consisting of Python with the Django framework for backend development, responsible for server-side logic and database management. The frontend will be developed using HTML, CSS, and JavaScript, with ECharts for creating data visualizations. The MySQL database will store the IMDB movie data and related information. Git will be used for version control, and GitHub will host the project and manage collaboration. Additionally, tools such as Pandas for data processing and pie charts for specific visualizations will be utilized to support the functionality of the platform.

# Related Work

**2.1 Chinese DOUBAN Movie Data Analysis Platform**

<https://www.bilibili.com/video/BV1A94y127qu/?spm_id_from=333.337.search-card.all.click&vd_source=369b00f99c15f6bb35635657e310a0c0>

This platform is a tool for detailed movie data analysis based on Douban(a popular Chinese movie and media rating platform) data.

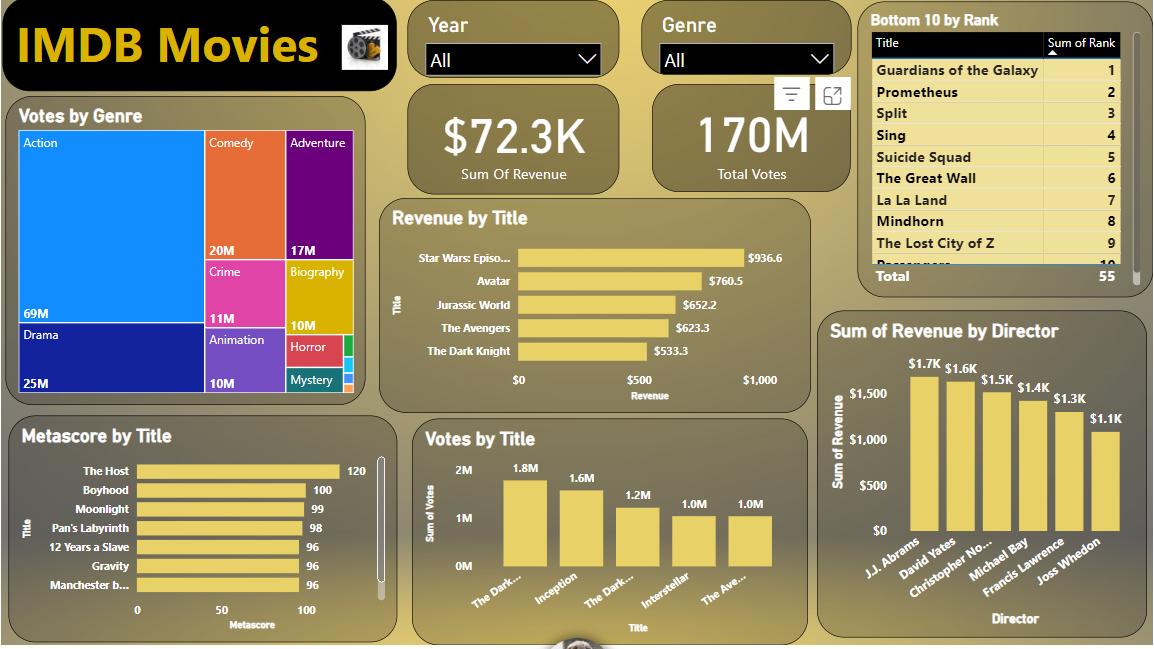


* **Overview**: This platform offers interactive pie charts, line graphs, and extensive movie tables, along with detailed analytics on movie genres, ratings, and production countries. Users can delve deeper into particular analysis by the categories on the left-hand sidebar, which include time analysis, map analysis, and word clouds for actor appearances and movie reviews.It provides insights into trends, ratings, and performance within the film business, with a special emphasis on Chinese-language productions.
* **Differences**: The Douban platform provides a structured way to visualize genres and ratings. However, our platform goes beyond this by offering more diverse visualizations such as ***box office performance***, ***movie duration and language diversity analysis***, ***sentiment analysis*** through word clouds, and ***customizable filters*** to focus on specific movie categories. Furthermore, our platform provides a global view, delivering data insights from film industries worldwide, rather than concentrating on a single country.

**2.2 IMDB Movies Dashboard with PowerBI**

<https://www.novypro.com/project/imdb-dashboard>

This platform is a PowerBI\_based movie data visualization tool designed to offer insights into various metrics from the IMDB database.

 FIGURE 2: PowerBI\_based IMDB Movies Dashboard

* **Overview**: The primary purpose of this dashboard is to visualize IMDb movie data. It provides information on votes based on genre and title, revenue for each movie, total revenue for each director, and meta score comparisons. It is built on PowerBI rather than developed from scratch.

* **Differences**: While the PowerBI dashboard gives a solid overview of movie performance based on ratings and revenue, our platform offers ***deeper insights*** with features like ***detailed box office and genre analysis***, ***movie duration and language diversity analysis***, and potential ***real-time data refresh*** capabilities. Additionally, we offer advanced visualizations, such as ***radar charts*** for genre performance and ***funnel charts*** for production countries. Most importantly, our platform is developed entirely from scratch.

**2.3 IMDB Dashboard Highlighting Jim Carrey based on Tableau**

[Lights, Camera, Action: Tableau and IMDb Feature Data Visualization Campaigns for Movie and TV Lovers - Salesforce](https://www.salesforce.com/news/stories/imdb-tableau-movie-visualizations/)

This Tableau-based data visualization platform offers a detailed exploration of Jim Carrey’s movie career.

* **Overview**: This platform offers a specialized analysis of Jim Carrey’s movie career, with a focus on IMDb ratings, audience votes, and comparisons with other comedic actors. It is ideal for both movie enthusiasts and professionals, offering insights into Carrey’s career and his standing among comedic legends. It is built on Tableau rather than developed from scratch.
* **Differences**: Our project encompasses the whole film business, enabling users to analyze ***patterns across genres, production regions, languages, and movie durations***, whereas this Tableau dashboard shines in examining the performance of a single actor. Additionally, if we are capable of completing desirable and optional features, we might be able to offer real-time updates and ***customizable data filters***. All of these would make our platform more versatile for examining patterns in the industry as a whole rather than just specific actors. Moreover, our platform is developed entirely from scratch which is different from this tableau dashboard.

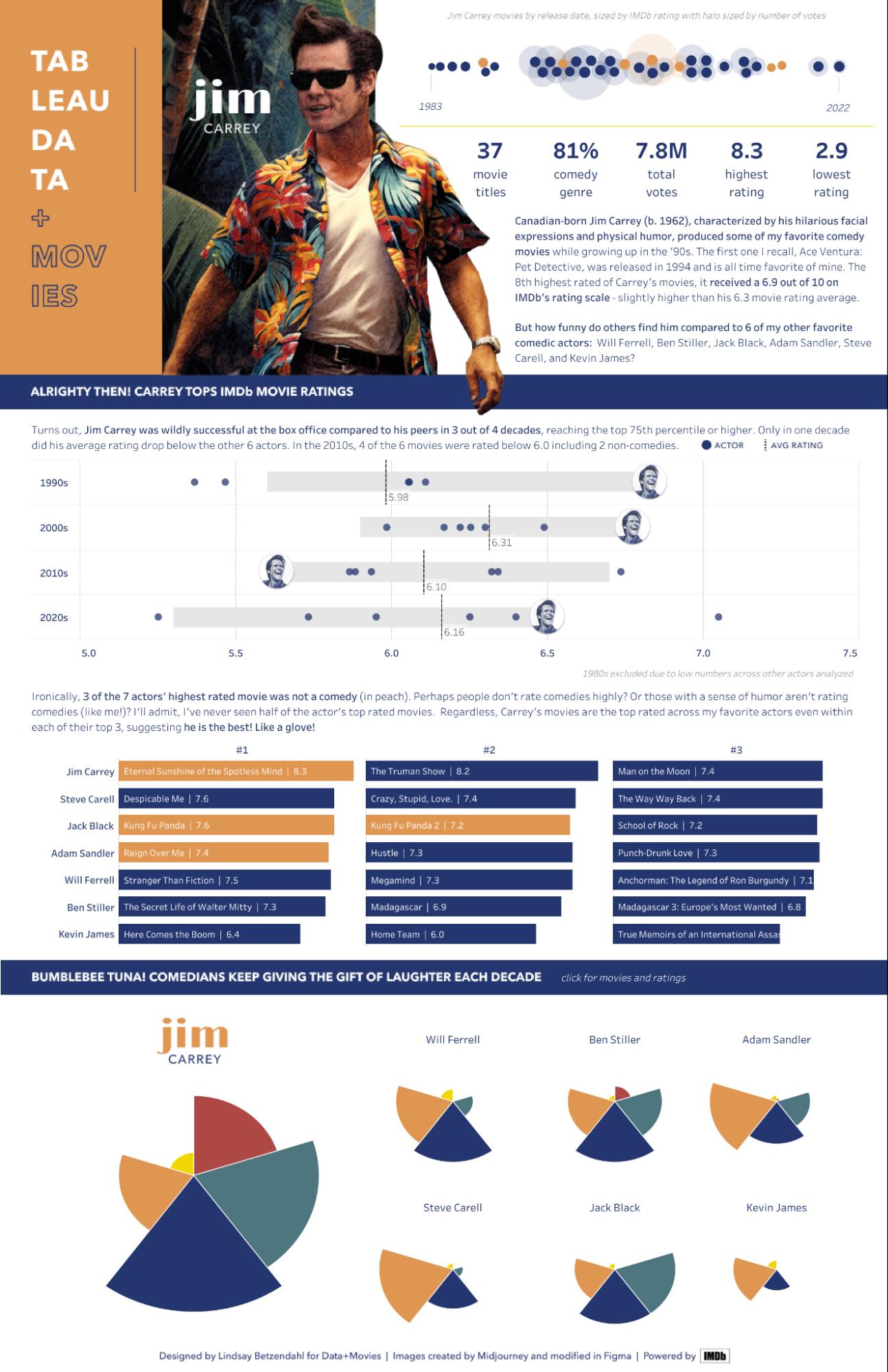


FIGURE 3: Tableau\_based IMDB Dashboard Highlighting Jim Carrey

# Proposed High level Requirements

* 1. **Functional Requirements**
     1. **Essential Features** (the core features that you definitely need to finish):

(For each essential features, please give a rough estimation in terms of person hours or an range of person hours)

1. Interactive Movie Data Visualization

* Description: As a data analyst, I want to explore visualizations of various movie attributes (ratings, genres, release years, production countries, etc.) using multiple chart types, so that I can quickly identify patterns and trends in the movie industry.
* Functionality Included:
  + Circular chart showing movie rating distribution.
  + Line chart comparing ratings to awards won.
  + Bar chart displaying the top 20 movie production companies.
  + Pie chart representing the distribution of movie release years by intervals.
  + Funnel chart showing the number of movies produced by country.
* Estimation: 25-35 person hours

1. Detailed Box Office and Genre Insights

* Description: As a studio executive, I want to visualize the relationship between movie ratings and box office performance, and analyze genre distribution, so that I can assess which movie types generate more revenue and which genres are most popular.
* Functionality Included:
  + Radar chart showing box office performance based on movie ratings.
  + Line chart visualizing the distribution of movies across different genres.
* Estimation: 15-25 person hours

1. Movie Duration and Language Diversity Analysis

* Description: As a media researcher, I want to explore the distribution of movie runtimes and the diversity of languages in movies, so that I can study trends in movie length and language use across the industry.
* Functionality Included:
  + Histogram showing the frequency distribution of movie runtimes.
  + Bar chart visualizing the number of movies produced in various languages.
* Estimation: 12-18 person hours
  + 1. **Desirable Features**:

1. Review Sentiment and Word Cloud Generation

* Description: As a film critic, I want to generate a word cloud from movie reviews and perform sentiment analysis, so that I can understand the general tone of user feedback and the most frequently mentioned terms in reviews.
* Functionality Included:
  + Word cloud generation based on user reviews.
  + Sentiment analysis of reviews.
* Estimation: 15-20 person hours

1. Customizable Data Filters

* Description: As a data scientist, I want to apply custom filters (e.g., rating, genre, language) to the visualized data, so that I can focus on specific movie categories or attributes relevant to my research.
* Estimation: 12-18 person hours
  + 1. **Optional Features**:

1. User Account and Preferences System

* Description: As a frequent user of the platform, I want to create an account and save my dashboard preferences, so that I can return to the platform with my customized settings intact.
* Estimation: 25-30 person hours

1. Real-Time Data Refresh

* Description: As a system administrator, I want the platform to refresh movie data in real-time from external sources, so that the visualizations always reflect the latest information without manual updates.
* Estimation: 20-25 person hours  
  1. **Nonfunctional Requirements**
     1. **Security requirements**

1. Role-Based Access Control: If a user account system is implemented, role-based access control must be enforced, ensuring that only administrators can perform tasks like data import and refresh, while regular users can access only general visualizations and reports.
2. Secure Database Management: The MySQL database must be protected against unauthorized access, with credentials securely stored using environment variables. Additionally, the platform should implement protections against SQL injection attacks.
3. API Security: Any external data retrieved (e.g., IMDB data) should be securely handled, with API keys stored in environment variables and not exposed in the codebase. Rate limits should be enforced to prevent API abuse.

# Management Plan

## Objectives and Priorities

1. Primary Objective: Successfully implement all essential features, including interactive movie data visualization, detailed box office insights, and movie duration/language diversity analysis.
2. Secondary Objective: Implement desirable features such as sentiment analysis and real-time data refresh, ensuring the platform remains scalable.
3. Quality Goals: Maintain high-quality software with no known critical bugs, ensuring user satisfaction and smooth functionality.
4. Deployment Goal: Successfully deploy the platform, ensuring it's accessible to the target audience (movie enthusiasts, data analysts, etc.).

## Risk Management

1. Loss of Team Members: The loss of a key team member can delay project progress and increase workload for remaining members.
2. Add New Team Members: Adding new members late in the project can disrupt team dynamics and require additional onboarding time.
3. Lack of Motivation or Responsibility: A lack of motivation among team members can lead to reduced productivity and slower progress.
4. Duplicate Work: Unclear task assignment can result in team members unintentionally working on the same components.
5. Worked on Wrong Components: Miscommunication or misunderstanding can lead to work being done on incorrect components.
6. Useless Work: Effort spent on unnecessary tasks due to unclear priorities or scope.
7. Inconsistent Work: Lack of collaboration and clear standards can result in inconsistent work quality across team members.
8. Unclear Requirements: Vague or incomplete requirements can cause confusion and delays in project development.
9. Scope Creep: The addition of new features beyond the initial project scope can overwhelm the team and extend deadlines.
10. Constant Requirements Changes: Frequent changes in requirements can disrupt progress and lead to project delays.
11. Improper Task Assignments: Assigning tasks without considering team members' skills can lead to inefficiency.
12. Improper Planning: Poor planning can result in missed deadlines and last-minute work.
13. Lack of Management Skills: Inexperienced leadership may struggle with task prioritization and resource allocation.
14. Not Familiar with the Framework Used: Team members may face delays due to a lack of knowledge about the chosen framework.
15. Not Familiar with the Programming Language Used: Lack of familiarity with the programming language can slow down development and increase errors.
16. Not Familiar with Unit Testing: Inexperience with unit testing can result in poor test coverage and undetected bugs.
17. Other Technology Incompetence: Lack of experience with essential tools or technologies can hinder progress.
18. Not Familiar with Git: Misuse of Git can lead to version control issues, code conflicts, and lost progress.
19. Improper Design: Flawed design decisions can lead to inefficient or incorrect implementation.
20. Improper Technology Stack: Choosing an inappropriate technology stack can result in performance issues and compatibility problems.
21. Messy Code: Poorly structured code can make maintenance and debugging difficult.
22. Not Enough Testing: Inadequate testing increases the risk of undetected bugs reaching production.
23. Not Enough Time for Integration and Deployment: Insufficient time for integration and deployment can result in a rushed and unstable product.

**Risk Management Sheet Link:** [**https://docs.google.com/spreadsheets/d/1bdQFEZOjxqHkdqhWfy8k1C0No-5Ga\_CytVS9Iqz8x5k/edit?gid=0#gid=0**](https://docs.google.com/spreadsheets/d/1bdQFEZOjxqHkdqhWfy8k1C0No-5Ga_CytVS9Iqz8x5k/edit?gid=0#gid=0)

## Timeline

| Iteration | Functional Requirements(Essential/Disable/Option) | Tasks (Cross requirements tasks) | Estimated/real person hours |
| --- | --- | --- | --- |
| Iteration 1 | Essential Features | Data import, basic visualizations setup | 80-100 hours |
| Iteration 2 | Desirable Features | Implement review sentiment analysis | 100 hours |
| Iteration 3 | Optional Features | User accounts and real-time data refresh | 110 hours |

# Configuration Management Plan

**1）Tools**

In this project, we will use the following tools for configuration management and development:

Git and GitHub: Used for version control and collaboration.

Visual Studio Code (VS Code): The primary Integrated Development Environment (IDE) for code editing.

GitHub Actions: For Continuous Integration (CI) and Continuous Deployment (CD) automation.

Docker: To containerize the application and ensure consistency across environments.

SonarQube: For static code analysis to ensure code quality and detect vulnerabilities.

Postman: For API testing and debugging.

**2） Code Commit Guideline and Git Branching Strategy**

Code Commit Guidelines:

Commit messages should follow a consistent format (e.g., [Feature], [Bugfix]), and each commit should include only relevant changes.

Regular, small, and functional commits are encouraged.

Git Branching Strategy: We will follow the Gitflow branching strategy:

Main Branch (main): Production-ready code only.

Develop Branch (develop): Integrates features and fixes for testing before release.

Feature Branches (feature/feature-name): Used for developing new features.

Hotfix Branches (hotfix/bug-description): Used for critical bug fixes.

Pull requests are required for merging, and peer review is mandatory before approval.

**3） CI/CD Plan**

Continuous Integration: Every code push or pull request will trigger automated workflows using GitHub Actions, running unit tests and static code analysis.

Continuous Deployment: Upon merging into main, the application will be built into a Docker container and deployed to the cloud (e.g., AWS, Heroku). Docker containers will ensure consistent environments between development and production.

Monitoring: Tools like Prometheus and Grafana will be used to monitor the health of the deployed application, with alerts sent through Slack in case of issues.

# Quality Assurance Plan

## Metrics

(Describe the metrics to be used in the project to measure the quality of your software. Each metric should be measurable and quantifiable. Examples of metrics include product complexity (LOC, # of files, # of classes, # methods, cyclomatic complexity, etc.) , defect rate (# of defect per KLOC), # of test cases, test case pass rate, cost (# of person hours used), # of user stories completed, etc. **The result of these metrics should be reported in the progress report/ iteration summary sheet.**)

| Metric Name | Description |
| --- | --- |
| Lines of Code (LOC) | The total number of lines written. |
| Cyclomatic Complexity | A metric to measure how complex your code is. |
| Defect Rate | Number of bugs per thousand lines of code. |
| Test Coverage | How much of your code is tested through unit tests. |
| Test Case Pass Rate | percentage of test cases that pass successfully. |

* 1. Coding Standard

(Describe any coding standard to be used)

All source code files, configuration files, documentation, and database contents must use **UTF-8** character encoding. UTF-8 is a widely used encoding standard that supports all Unicode characters, including Chinese, Arabic, special symbols, etc. Using UTF-8 ensures cross-platform compatibility and globalization support, avoiding encoding errors or character display issues.

Variable names, function names, and class names should follow **camelCase** for variables and functions, and **PascalCase** for class names. This enhances code readability and adheres to industry best practices.

All code must use consistent indentation (e.g., 4 spaces per indentation level).

Functions or methods should be kept as short as possible to maintain single responsibility and avoid excessive complexity.

## Code Review Process

(Everyone should review all documents to be submitted. Here you will mainly describe how the code review will be done. Who will review the code, e.g. design or implementation leader will review all code or team members review each other’s code. Do you use pull requests for the code review? Is there a checklist to help review? What feedback should the reviewer provide?)

1. **Who Will Review**:Code reviews will be conducted through a **peer-review system**, where team members review each other’s code. This ensures shared ownership of the codebase and promotes knowledge sharing within the team.

For critical components or high-complexity modules, the **implementation leader** or **design lead** will perform an additional review to verify architectural integrity and design consistency.

1. **Code Review Workflow**:**Pull Requests (PRs)** will be used for all code submissions. Developers must submit a pull request for their feature branch before merging into the main or development branch.

All pull requests will require at least **two approvals** from other team members before the code can be merged. The implementation or design leader must approve complex or high-impact changes.

1. **Review Checklist**: Reviewers will use a standardized checklist to ensure consistency and thoroughness during the review process. The checklist includes:

* **Functionality**: Does the code implement the required functionality? Does it pass all defined unit tests?
* **Code Quality**: Is the code clean, readable, and maintainable? Is it following the defined coding standards (e.g., variable naming, code formatting)?
* **Error Handling**: Is error handling implemented properly? Does the code account for edge cases and potential exceptions?
* **Performance**: Are there any performance concerns? Is the algorithm efficient and scalable?
* **Security**: Are there any potential security vulnerabilities or concerns, such as improper input validation or insecure data handling?
* **Documentation**: Are comments and documentation adequate? Does the code provide enough context for future maintainers?

1. **Feedback**:Reviewers should provide **constructive feedback** that focuses on improving the quality of the code. Feedback should:

* Highlight **issues** such as bugs, potential improvements in logic, or deviations from coding standards.
* Offer **suggestions** for refactoring or improving the clarity and efficiency of the code.
* Acknowledge **positive aspects** of the code, such as well-written tests or clean design patterns.

All feedback should be clear, actionable, and considerate, fostering a positive and productive review environment.

1. **Post-Review Process**:After feedback is given, the developer must make the necessary changes and submit the updated code for re-review.

The reviewer must verify that all issues have been addressed before giving final approval.

Once the code passes all reviews, it can be merged into the main branch and marked as complete.

## Testing

(Both manual testing and automated testing should be considered. Both unit testing and integration testing should be considered. Briefly describe the testing tools/framework to be used, the personnel involved (e.g. the QA leader will focus on the integration testing and each developer will unit test their own code), when and what types of testing will be performed, the testing objectives, etc)

**Types of Testing**:

**Unit Testing**: Each developer will be responsible for writing and executing unit tests for the code they develop. Unit tests will verify that individual functions, classes, or modules work as expected in isolation. This ensures that the smallest pieces of the software are functioning correctly.

**Integration Testing**: The **QA leader** will focus on integration testing to verify that different components of the system interact properly. Integration tests will be performed after unit tests to ensure the integration of modules works seamlessly without introducing any bugs.

**Manual Testing**: QA team members will perform manual testing to validate the overall user experience and ensure that the software meets the specified requirements. This will include testing user interfaces, workflows, and edge cases that may not be covered by automated tests.

**Testing Tools/Frameworks**:

**Integration Testing**:

Integration testing will be automated using tools like **Selenium** (for UI integration testing) or **Postman** (for API testing).

**Automated Testing**:

**Selenium** and **Cypress** will be used for end-to-end testing of web applications, focusing on ensuring the system’s UI works as expected.

**Jenkins** or another CI/CD tool will be used to automate the running of tests upon every new code commit (continuous integration).

**Manual Testing**:

Manual test cases will be tracked using tools like **Jira** or **TestRail** to ensure proper coverage and easy tracking of issues.

**Personnel Involved**:

**QA Leader**:

The QA leader will focus on integration testing and coordinate all testing efforts, ensuring proper test coverage and handling the more complex testing scenarios such as system-wide integrations and end-to-end testing.

**Developers**:

Each developer will be responsible for writing unit tests for their code and performing initial testing before submitting the code for review. They will also assist the QA team in fixing any issues found during testing.

**QA Team**:

The QA team will handle manual testing and ensure that the software meets the acceptance criteria. They will also write automated test scripts for regression and integration testing.

**When Testing Will Be Performed**:

**Unit Testing** will be performed continuously during the development phase, with each developer responsible for testing their own code.

**Integration Testing** will be performed after unit tests are passed, ensuring the smooth interaction between different modules.

**Manual Testing** will be conducted during the QA phase before each release, focusing on user interfaces and overall functionality.

## Defect Management

(Describe the tool to be used to manage the defect (e.g github issues). The types of defects to look at. The actions or personnel for defect management. )

We will use **GitHub Issues** as the primary tool for defect tracking and management. GitHub Issues provides an efficient platform for reporting, categorizing, and prioritizing defects. Each defect will be logged as an issue, complete with details such as the severity, steps to reproduce, and any relevant screenshots or error logs.

**Types of Defects**: The types of defects we will track include:

**Functional Defects**: Issues where the software does not behave as expected according to the requirements or design specifications. These include incorrect outputs, broken workflows, or missing functionality.

**UI/UX Defects**: Visual or usability issues that affect the user experience. These include problems like misaligned elements, incorrect colors, or non-responsive design.

**Security Defects**: Vulnerabilities that could expose the system to attacks or data breaches. This includes issues like improper input validation, weak authentication mechanisms, or improper encryption.

**Compatibility Defects**: Problems arising from the software not functioning correctly across different platforms, browsers, or devices.

**Regression Defects**: Bugs introduced in previously working functionality after new changes or features are added.

**Personnel for Defect Management**:

**Developers**: Responsible for fixing defects that are assigned to them and ensuring that the code changes do not introduce new bugs.

**QA Leader/Team**: The QA leader will prioritize defects and manage the overall defect resolution process. The QA team will re-test resolved defects to ensure the fixes are valid.

**Project Manager**: Responsible for tracking defect metrics (e.g., number of open defects, average time to resolve, etc.) and ensuring the defect backlog is properly managed and aligned with release goals.

**Testers**: Testers will continuously report any new defects during the testing phases and monitor the status of reported issues.

# References

(For more details, please refer to the encounter example in the book or the software version of the documents posted on blackboard. )

# Glossary

(Any acronym used in the document should be explained here)