TECHNICAL DOCUMENTATION

**Project Title :** AI-Powered Story Generation and Narration System

**Author :** Dayakar Vallepu

**Date :** 16 June 2025

**Organization:** AISPRY Pvt. Ltd.

**Introduction**

* This document provides a technical overview of the AI-Powered Story Generation and Narration System, a full-stack application leveraging generative AI (Gemini 1.5 Flash, Mistral) to create personalized, age-appropriate stories with multilingual support and text-to-speech (TTS) narration.

**Problem Statement**

* Traditional storytelling tools lack personalization, interactivity, and accessibility, especially for children and teens. There is no widely available mobile solution that allows users to generate custom stories with engaging, age-appropriate narration using AI.
* **Objective:**
* Maximize user engagement and app retention.
* **Constraints:**
* Minimize churn
* **Success criteria:**
* **Business Success Criteria:**

Achieve 70%+ monthly user retention within 12 months.

* **ML Success Criteria:**

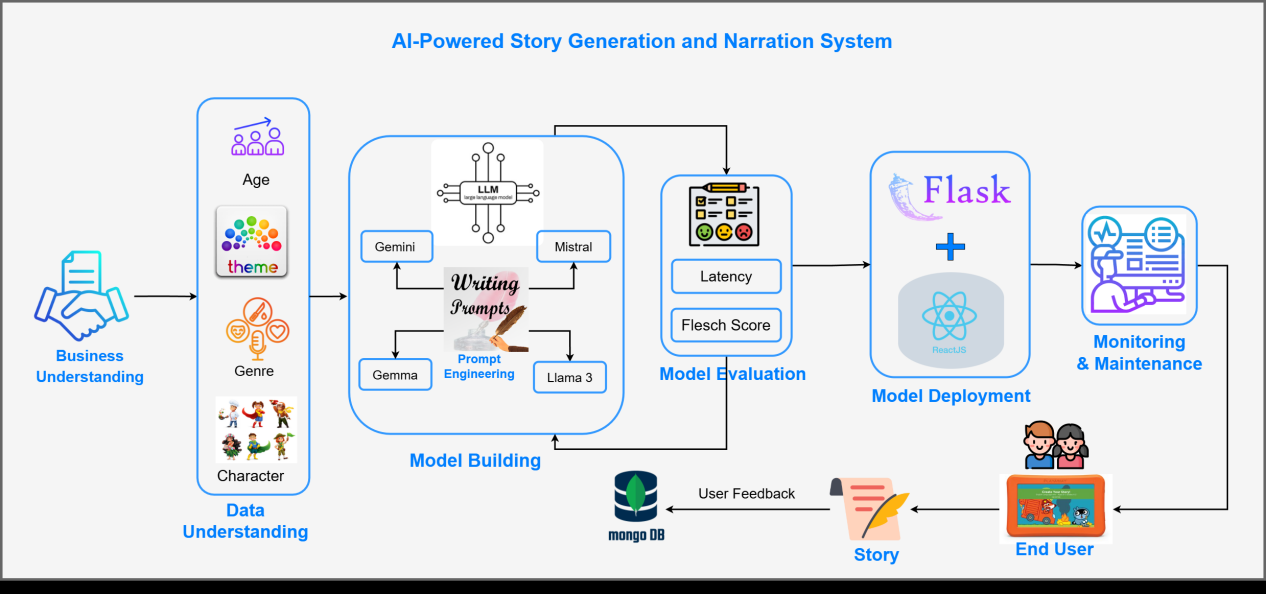
Generate coherent, age-appropriate stories with ≥90% user approval.

* **Economic Success Criteria:**

10,000 active users and positive unit economics within 12 months.

* Scope:
* Real-time story generation using LLMs (e.g., Gemini, Mistral)
* Frontend-backend integration
* MongoDB data management
* Evaluation and monitoring systems

**System Architecture**



* The system is built on a full-stack architecture:
  + Frontend: React.js
  + Backend: Flask (Python)
  + AI Engine: Gemini 1.5 Flash (Cloud) + Mistral (Local)
  + Database: MongoDB
  + TTS: Google Cloud TTS, ElevenLabs API

**Data Understanding**

The AI-Powered Story Generation and Narration System primarily relies on user-driven input rather than traditional datasets. Since this project employs prompt-based generation using LLMs instead of training models on labeled data, the data understanding phase involved defining, organizing, and validating user input fields used for generating personalized stories.

* **Data Sources** : -

| Input Field | Description |
| --- | --- |
| Age Group | Age range of the target user (e.g., 3–8, 9–15, etc.) |
| Genre | Story type (e.g., comedy, adventure, mystery) |
| Theme | Moral or message (e.g., honesty, bravery, friendship) |
| Main Characters | Custom characters input by the user |
| Language | Desired output language (e.g., English, Telugu, Hindi) |

* **Data Mapping and Constraints**

Inputs were mapped manually to enforce constraints, readability standards, and story structure:

| Age Group | Target FRE Score Range | Readability Level |
| --- | --- | --- |
| 3–8 | 80–100 | Easy |
| 9–15 | 60–80 | Moderate |
| 16–19 | 50–60 | Challenging |
| 20+ | 30–50 | Advanced |

* **Data Dictionary**

A data dictionary was created to validate and structure user inputs before prompt generation.

| Field | Type | Example | Constraints |
| --- | --- | --- | --- |
| age\_range | String | "9-15" | Must match allowed ranges |
| genre | String | "Adventure" | Limited to defined genres |
| theme | String | "Honesty" | Must be meaningful and child-appropriate |
| characters | List/String | ["Ali", "Zebra"] | Max 5 characters |
| language | String | "Telugu" | Optional |

* **Data Flow Diagram**

[User Input (Web UI)]

↓

[Input Validator & Mapper (Flask)]

↓

[Prompt Generator → LLM → Story Output]

↓

[Evaluation: FRE Score, Latency]

↓

[Storage in MongoDB (Story + Metadata)]

**Model Building**

The system uses prompt-based story generation leveraging large language models (LLMs). No traditional training was done. Instead, model building focused on:

· Selecting and integrating powerful LLMs (cloud + local)

· Designing dynamic prompts

· Validating outputs using readability metrics

* **Models Used**

| Model | Type | Usage |
| --- | --- | --- |
| Gemini 1.5 Flash | Cloud | Primary model via API for creative and accurate story generation |
| Mistral | Local (Ollama) | Fallback for offline or quota-restricted use |
| Gemma / LLaMA 3 | Local (Ollama) | Additional support for performance and experimentation |

### **Prompt Engineering (Core Logic)**

The prompt includes user inputs (age, genre, theme, characters, language) and ensures story readability using Flesch Reading Ease (FRE) constraints.

**Prompt Template**

prompt = f"""

    Write an imaginative and age-appropriate story for Indian children aged {age\_range}.

    Requirements:

    - Genre: {genre}

    - Main Characters: {characters}

    - Theme: {theme}

    - Story Length: Maximum 350 words.

    - Use simple, clear English vocabulary and sentence structure that is suitable for children aged {age\_range}.

    - The story MUST be written so that its Flesch Reading Ease (FRE) score is between {flesch\_score}.

    - The Flesch score requirement applies ONLY to the English story.

    - DO NOT include any translation, non-English words, or translation labels in the English story section.

    - Avoid complex words and long sentences for younger ages; use more advanced language for older ages.

    - Include a catchy, relevant title at the beginning.

    - End with a moral in the format: Moral: [your moral]

    - \*\*Do NOT add any label like 'English Story:' or similar. Only use 'Title:', the story text, and 'Moral:' as shown below.\*\*

    Format:

    Title: [Your title]

    [Story text]

    Moral: [your moral]

    """

    if age\_range in ["3-5", "3-8"]:

        prompt += (

            "\n- IMPORTANT: The story MUST have a Flesch Reading Ease (FRE) score between 80 and 100."

            "\n- Use normal words and keep medium sentences (6-9 words each)."

            "\n- Avoid any very easy or very complex vocabulary."

            "\n- Imagine you are writing for a 3–8 year old who is just learning to read."

            "\n- If the story is  very easy or more difficult, REWRITE it until it fits the FRE score range."

            "\n- Do NOT write a story that is outside this FRE score range."

            "\n- If you cannot write a story within this FRE range, DO NOT RETURN ANY STORY."

        )

    elif age\_range in ["9-15"]:

        prompt += (

            "\n- IMPORTANT: The story MUST have a Flesch Reading Ease (FRE) score between 60 and 80."

            "\n- The FRE score must NEVER be above 80 or below 60 for this age group."

            "\n- Use simple and clear words."

            "\n- Keep sentences short (10–14 words) and easy to understand."

            "\n- Avoid difficult vocabulary and long sentences."

            "\n- Do not use advanced or academic words."

            "\n- Imagine you are writing for a school student aged 9 to 15."

            "\n- If the story is too easy or too hard, or if the FRE score is outside 60–80, REWRITE it until it fits the FRE score range."

            "\n- You absolutely MUST NOT write a story with a Flesch score above 80 or below 60."

            "\n- Do NOT write a story that is outside this FRE score range."

            "\n- If you cannot write a story within this FRE range, DO NOT RETURN ANY STORY."

            "\n- EVEN WHEN TRANSLATING TO ANOTHER LANGUAGE, NEVER GO OUTSIDE THE FRE SCORE RANGE OF 60–80."

            "\n- Repeat: The story (and any translation) MUST have a Flesch Reading Ease (FRE) score between 60 and 80."

        )

    elif age\_range in ["16-19"]:

        prompt += (

            "\n- IMPORTANT: The story MUST have a Flesch Reading Ease (FRE) score between 50 and 60."

            "\n- Use clear language with some moderately advanced vocabulary."

            "\n- Keep most sentences between 10 and 16 words."

            "\n- Mix simple and moderately complex sentences, but avoid very long or academic sentences."

            "\n- Do not use too many advanced words."

            "\n- Write as you would for a high school student aged 16 to 19."

            "\n- If the story is too easy or too hard, REWRITE it until it fits the FRE score range."

            "\n- Do NOT write a story that is outside this FRE score range."

            "\n- If you cannot write a story within this FRE range, DO NOT RETURN ANY STORY."

        )

    elif age\_range in ["20+"]:

        prompt += (

            "\n- IMPORTANT: The story MUST have a Flesch Reading Ease (FRE) score between 30 and 50."

            "\n- Aim for a FRE score between 40 and 45. Do NOT write a story with a FRE score below 35 or above 45."

            "\n- Use advanced vocabulary, medium sentences, and  complex sentence structures."

            "\n- Do not simplify the language. Write as you would for college students or adults."

            "\n- If the story is very too easy (FRE > 50) or very too hard (FRE < 30), REWRITE it until it fits the FRE score range."

            "\n- Do NOT write a story that is outside this FRE score range."

            "\n- If you cannot write a story within this FRE range, DO NOT RETURN ANY STORY."

            "\n- If you do not follow the FRE rule, your answer will be rejected and regenerated."

        )

    if language and language.lower() != "none":

        lang\_key = language.lower()

        labels = label\_map.get(lang\_key, {

            "title": "Title",

            "story": "Story",

            "moral": "Moral"

        })

        prompt += f"""

After you have finished the English story above, translate ONLY the English story and its moral into {language} for Indian children.

\*\*Translation Instructions:\*\*

- Do NOT translate word-for-word. Use natural, fluent, and child-friendly {language} as used in everyday conversation.

- Ensure the translated story is easy for children in the target age group to understand.

- Use age-appropriate vocabulary and grammar for {language}.

- Do NOT include any English words unless they are proper nouns.

- Write ONLY the translated version, using these labels (translated in {language}):

    - {labels['title']}: [Translated title]

    - {labels['story']}: [Translated story]

    - {labels['moral']}: [Translated moral]

- DO NOT repeat the English story or moral.

- DO NOT include any English text in this section (except proper nouns).

- Structure:

    {labels['title']}: [Translated title]

    {labels['story']}: [Translated story]

    {labels['moral']}: [Translated moral]

- Even when translating to {language}, ensure the story would have a Flesch Reading Ease (FRE) score in the same range as required for English. Do NOT make the translation easier or harder than the English version. Do NOT go outside the FRE range for the selected age group, even in translation.”””

* **Local Model Integration (Ollama)**

Step 1: Install Ollama

For Windows/Mac/Linux

Visit: [https://ollama.com](https://ollama.com" \t "_new)

**Bash**

**# Linux/macOS (with curl)**

**curl -fsSL https://ollama.com/install.sh | sh**

**# Windows: Download the EXE and follow setup**

Step 2: Pull Models Locally

**Bash**

**ollama pull mistral**

**ollama pull gemma:2b**

**ollama pull llama3**

Step 3: Run Ollama Server

Bash

**ollama run mistral**

You’ll get an interactive prompt. Or for API use:

4. Gemini API Setup (Google Generative AI)

Step 1: Sign Up for API Access

Go to: <https://makersuite.google.com/app>

* Sign in with Gmail
* Click on Get API Key
* Copy the key

Step 2: Install SDK

Bash

**pip install google-generativeai**

Step 3: Use Gemini 1.5 Flash via API

**Python**

**import google.generativeai as genai**

**genai.configure(api\_key="YOUR\_GEMINI\_API\_KEY")**

**model = genai.GenerativeModel('gemini-1.5-flash')**

**response = model.generate\_content(prompt)**

**print(response.text)**

**Input Labels:**

**Age, Genre, Character, Theme**

| **Age Range** | **Label** | **Target Flesch Score** | **Reading Difficulty** |
| --- | --- | --- | --- |
| **3-8** | **Early Readers** | **80–100** | **Easy** |
| **9–12** | **Pre-teens** | **70–80** | **Fairly Easy** |
| **13–15** | **Young Teens** | **60–70** | **Standard** |
| **16–18** | **Older Teens** | **50–60** | **Fairly Difficult** |
| **19+** | **Adults** | **30–50** | **Difficult** |

**Evaluation**

The evaluation phase focused on verifying the quality, readability, performance, and structure of generated stories. Since no supervised training was performed, traditional ML evaluation metrics were replaced by content-specific quality checks.

* **Evaluation Goals**
* Ensure readability using Flesch Reading Ease (FRE)
* Maintain story structure (Title, Body, Moral)
* Minimize generation latency
* Collect and analyze user feedback
* Track issues using a custom Performance & Suitability Index (PSI)
* **Readability Assessment — Flesch Reading Ease (FRE)**

The FRE score is a standard readability formula used to assess whether generated stories are appropriate for the target age group.

**Formula:**

**FRE=206.835−(1.015×ASL)−(84.6×ASW)**

**Where:**

ASL = Average Sentence Length (words per sentence)

ASW = Average Syllables per Word

#### Implemented using textstat Python Library:

python

from textstat import flesch\_reading\_ease

score = flesch\_reading\_ease(story\_text)

* **Performance Metrics**

| Metric | Target | Tools |
| --- | --- | --- |
| Latency | ≤ 15 seconds | Timestamp tracking in Flask |
| Story Length | 250–350 words | Word count validator |
| FRE Score | Must be in age-appropriate range | textstat |
| PSI Score | Should be < 50 for healthy output | Custom formula |

### **Performance & Suitability Index (PSI)**

A composite score to measure story suitability and trigger drift warnings

Formula:

PSI=(0.3×Speed)+(0.4×FRE)+(0.3×LengthNormalized)

PSI > 50 → Flagged as drift in MongoDB (psi\_warning\_collection)

* **Challenges and Mitigation**

| Challenge | Mitigation |
| --- | --- |
| LLMs generating stories outside FRE range | Prompt constraints + regeneration logic |
| Subjective quality of morals | Manual reviews + user feedback |
| Latency spikes from Gemini | Fallback to local models (Mistral) |
| Format deviation (missing Title/Moral) | Regex parsing + structure enforcement |

**Deployment**

The AI-powered story generation system is deployed as a full-stack application using Flask (backend), React.js (frontend), and MongoDB (database). It supports both cloud-based LLMs (Gemini 1.5 Flash) and local models (Mistral, Gemma, LLaMA via Ollama) with Text-to-Speech integration.

* **Technology Stack Overview**

| Layer | Tools/Technologies |
| --- | --- |
| Frontend | React.js, HTML, CSS, JS |
| Backend | Flask (Python) |
| Database | MongoDB (local or Atlas) |
| LLMs | Gemini 1.5 Flash (cloud), Mistral/Gemma (local via Ollama) |
| TTS | Google Cloud TTS, ElevenLabs API |
| Monitoring | Power BI (optional) |

* **Backend Setup (Flask)**

**Prerequisites**

Python 3.10+

**requirements.txt** file with dependencies like Flask, pymongo, textstat

**bash**

**# 2. Create virtual environment**

**python -m venv venv**

**source venv/bin/activate # Windows: venv\Scripts\activate**

**# 3. Install dependencies**

**pip install -r requirements.txt**

**# 4. Add Gemini API key and Mongo URI in config.py**

**# Example config.py**

**GEMINI\_API\_KEY = "your\_api\_key"**

**MONGO\_URI = "mongodb://localhost:27017/"**

**Run the Flask Server**

**bash**

**python app.py**

**Server runs at: http://localhost:5000**

* **Frontend Setup (React.js)**

**Prerequisites**

**Node.js & npm (Download from <https://nodejs.org)>**

**Steps**

**bash**

**# 1. Go to frontend folder**

**cd storygen/frontend**

**# 2. Install dependencies**

**npm install**

**# 3. Start development server**

**npm start**

**Runs at: http://localhost:3000**

**Connect to Flask Backend**

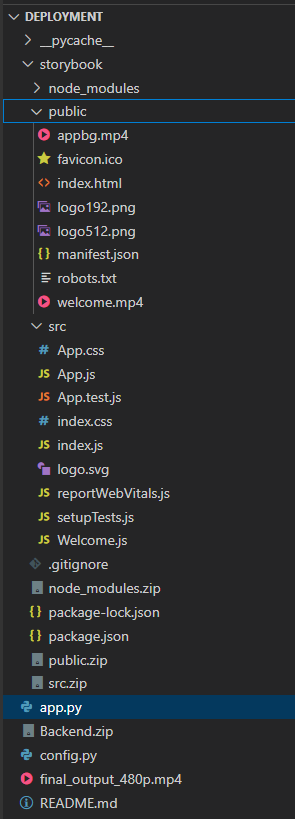
React fetches from Flask at http://localhost:5000. Ensure CORS is handled in Flask:

**python**

**from flask\_cors import CORS**

**CORS(app)**

* **Final Folder Structure**

****

**Power BI Integration and Monitering**

To monitor content quality, system performance, and user engagement, Power BI was used for building interactive dashboards. It provides real-time visualization of FRE scores, latency, PSI drift, and user feedback.

**Objectives**

* Track story readability by age group
* Monitor average latency across themes/genres
* Detect content drift using PSI thresholds
* Analyze user ratings and feedback

**MongoDB to Power BI Integration**

#### Required Tools:

* MongoDB (local or Atlas)
* Python (for ETL)
* Power BI Desktop

Repeat this for other collections like psi\_warning\_collection or login\_logs\_collection.

**Load CSV into Power BI**

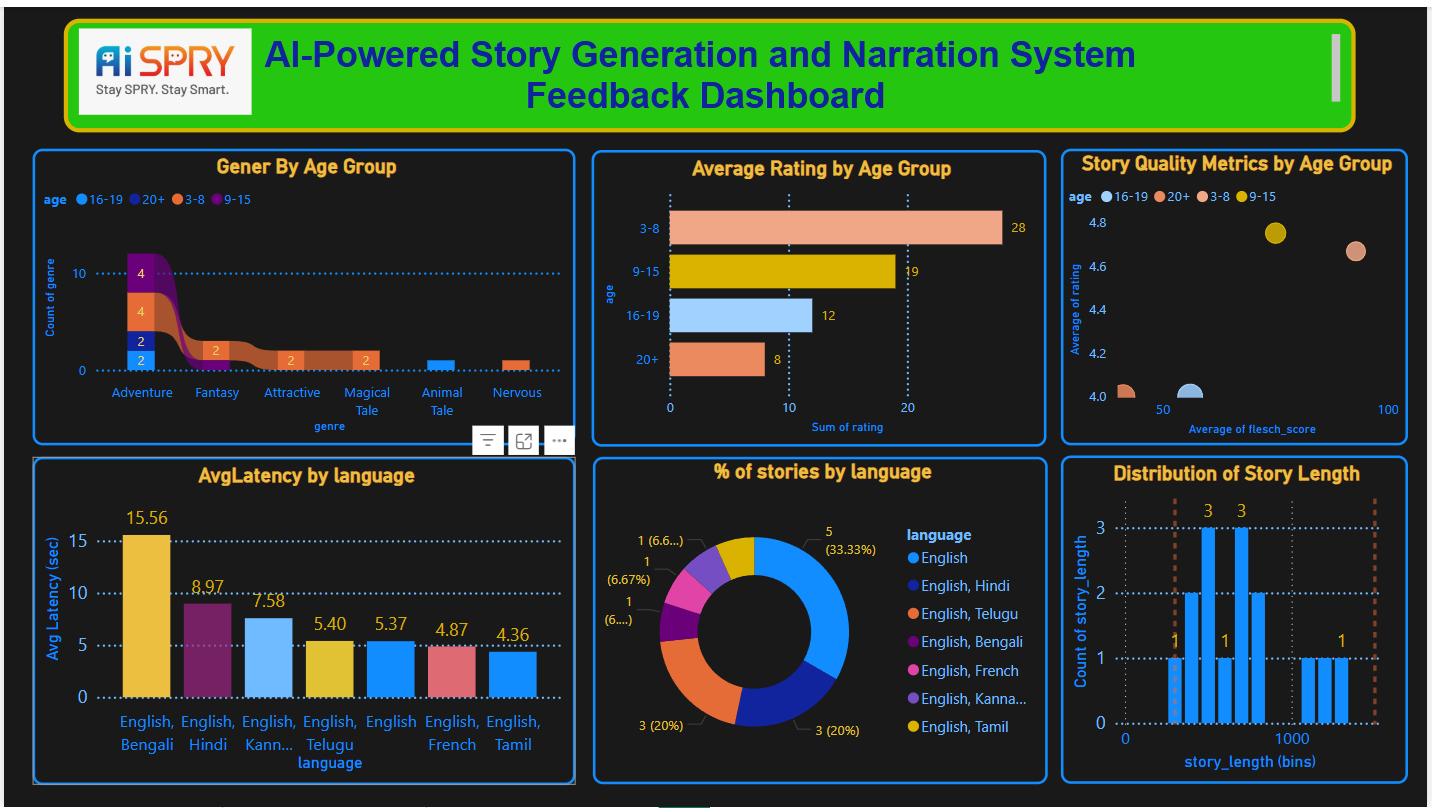
Open Power BI Desktop

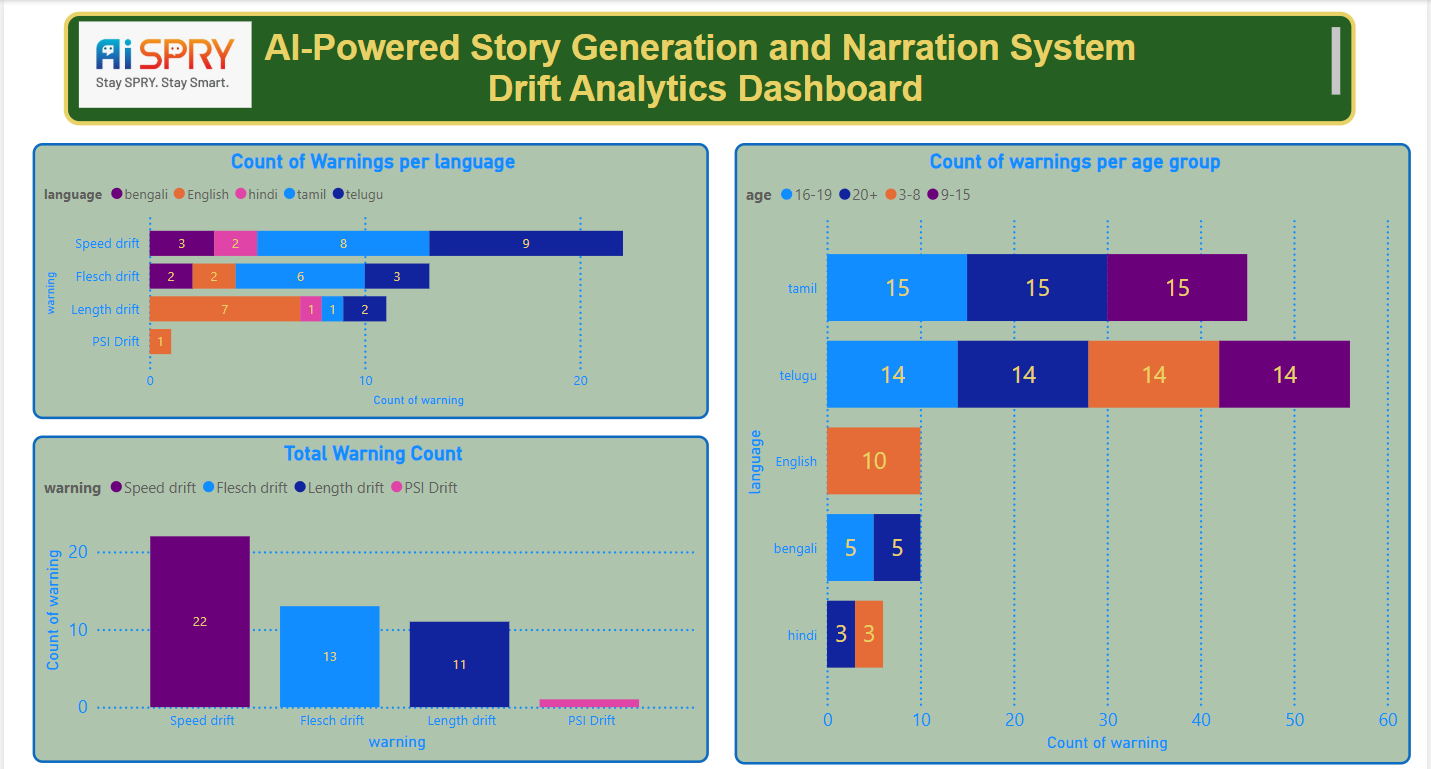
Click “Get Data” → “Text/CSV”

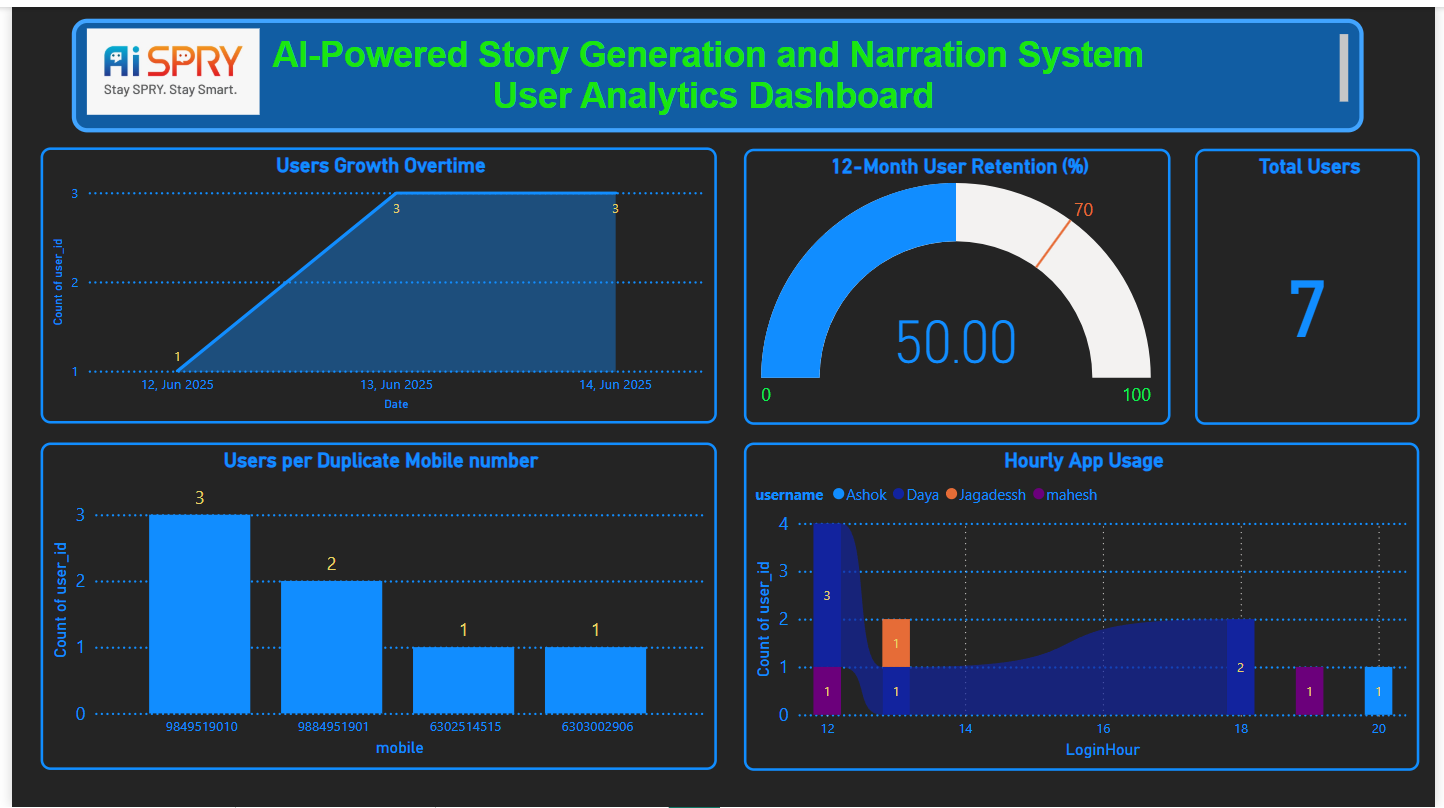
Load feedback\_data.csv, psi\_data.csv, etc.

Click “Transform Data” to clean and format fields

**Key Dashboards & Visuals**







**ETL Script (Python + PyMongo → CSV)**

**Python:**

**from pymongo import MongoClient**

**import pandas as pd**

**# Connect to MongoDB**

**client = MongoClient("mongodb://localhost:27017/")**

**db = client["storygen\_db"]**

**# Example: Load feedback collection**

**collection = db["feedback\_collection"]**

**data = list(collection.find())**

**df = pd.DataFrame(data)**

**# Clean up and export to CSV**

**df.drop(columns=["\_id"], inplace=True, errors="ignore")**

**df.to\_csv("feedback\_data.csv", index=False)**

**To achieve and visualize 70%+ monthly user retention within 12 months using Power BI Gauge Chart, follow this complete step-by-step implementation using your login logs.**

**Goal:**

**Use a Gauge chart to track and visualize the percentage of users retained monthly for the past 12 months with a target of ≥ 70%.**

**STEP 1: Load Data into Power BI**

**Open Power BI.**

**Click Home → Get Data → Text/CSV.**

**Load your login data CSV.**

**In Power Query, ensure your data looks like:**

**user\_id datetime username password device\_type**

**STEP 2: Format the Date Column (Power Query)**

**In Power Query:**

**Right-click on datetime → Change Type → Date/Time**

**Create a new column (optional but preferred for clean logic):**

**Add Column → Date → Date Only**

**Name it login\_date**

**Click Close & Apply**

**STEP 3: Create Calculated Table – UserFirstLogin**

**Go to Modeling → New Table:**

**dax**

**UserFirstLogin =**

**SUMMARIZE(**

**LoginData,**

**LoginData[user\_id],**

**"FirstLogin", MIN(LoginData[login\_date])**

**)**

**STEP 4: Create Retention Mapping Table**

**This maps every login within 12 months of first login.**

**Go to Modeling → New Table:**

**dax**

**RetentionMapping =**

**GENERATE(**

**SELECTCOLUMNS(**

**UserFirstLogin,**

**"uf\_user\_id", [user\_id],**

**"FirstLogin", [FirstLogin]**

**),**

**FILTER(**

**LoginData,**

**LoginData[user\_id] = [uf\_user\_id] &&**

**DATEDIFF([FirstLogin], LoginData[login\_date], MONTH) <= 11**

**)**

**)**

**STEP 5: Add MonthDiff Column**

**Go to RetentionMapping → New Column:**

**dax**

**MonthDiff = DATEDIFF([FirstLogin], [login\_date], MONTH)**

**STEP 6: Create Retention Measure**

**Go to Modeling → New Measure:**

**dax**

**Retention12MonthPercent =**

**VAR BaseUsers =**

**CALCULATE(**

**DISTINCTCOUNT(RetentionMapping[uf\_user\_id]),**

**RetentionMapping[MonthDiff] = 0**

**)**

**VAR RetainedUsers =**

**CALCULATE(**

**DISTINCTCOUNT(RetentionMapping[uf\_user\_id]),**

**RetentionMapping[MonthDiff] = 11**

**)**

**RETURN**

**DIVIDE(RetainedUsers, BaseUsers, 0) \* 100**

**STEP 7: Create the Gauge Chart**

**In Report View, select the Gauge visual.**

**Assign fields:**

**Value → Retention12MonthPercent**

**Target value → 70**

**Min → 0**

**Max → 100**

**Show data label as percentage**

**Result**

You’ll see a gauge showing % of users retained after 12 months.

For it to work:

You must have:

At least some users with MonthDiff = 11

First logins (MonthDiff = 0)

If your dataset has only recent dates (like June 2025), you won’t see any MonthDiff = 11. Use sample data or wait until data accumulates.

**Challenges**

**Data Understanding**

| Challenge | Mitigation |
| --- | --- |
| No publicly available datasets | Used user-input-driven system and created a data dictionary manually |
| Mapping age groups to readability (FRE) | Created custom FRE brackets using multiple academic sources |
| Ambiguous user inputs (e.g., complex characters or themes) | Used dropdowns and validation at frontend to limit inputs |

**Model Building (Prompt Engineering)**

| Challenge | Mitigation |
| --- | --- |
| FRE score not matching age group | Enforced FRE range directly inside the prompt + regenerated up to 3 times |
| Output format inconsistent (Title, Story, Moral missing) | Strict regex-based parsing and clear prompt structure instructions |
| Gemini API rate limits / latency | Integrated fallback models (Mistral, Gemma via Ollama) |
| Language translation drift (loss of meaning or readability) | Used language-specific labels and added FRE-based constraints in translated prompts |
| Incoherent or weak morals in stories | Reinforced age-specific moral clarity with strict prompt guidance |

**Deployment**

| Challenge | Mitigation |
| --- | --- |
| Gemini API rate limiting during peak hours | Added secure API key management with retry logic + offline model fallback |
| Latency beyond 15 sec during high load | Monitored generation time and routed to local models if delay is detected |

**Monitoring & Maintenance**

| Challenge | Mitigation |
| --- | --- |
| Hard to track which stories failed or drifted | Stored drift logs (PSI, FRE violations) in psi\_warning\_collection |
| Limited observability of user behavior | Integrated Power BI for feedback, FRE score trends, and usage analytics |
| Need for modular upgrades (e.g., mobile app later) | Designed API and frontend as loosely coupled components for scalability |

* **Future Enhancements**
* Docker containerization
* CI/CD integration
* Mobile app version
* Image/video generation (multimodal)
* Advanced voice synthesis
* **Contact**

**Author**: Dayakar Vallepu

**Email**: [dayavallepu@gmail.com](mailto:dayavallepu@gmail.com)

**Organization**: AISPRY Pvt. Ltd.