```python
# Judge LLM Evaluation Notebook with All Advanced Features
# Requirements: Gemini 1.5 Pro API key, .png chart files, insight text files

import os
import csv
import re
from PIL import Image
import google.generativeai as genai


# === CONFIGURATION ===
# Set your API key and paths for chart images, insight drafts, and output logs
API_KEY = "YOUR_API_KEY"  # Replace with your actual Gemini API key
CHART_FOLDER = "charts"
INSIGHT_FOLDER = "insights"
OUTPUT_CSV = "judge_eval_results.csv"
N_VARIANTS = 3  # Number of insight drafts to evaluate per chart for reranking


# === SETUP ===
# Configure Gemini API and instantiate the Gemini 1.5 Pro model
genai.configure(api_key=API_KEY)
model = genai.GenerativeModel("gemini-1.5-pro")


# === PROMPT TEMPLATE ===
# Builds a consistent prompt with rubric instructions for the Judge LLM
# Takes insight text and instructs the model to evaluate across 4 dimensions
# Includes a fail-safe message if the insight violates MTSS alignment rules

def build_prompt(insight_text):
    return f"""
You are an education policy expert reviewing an insight based on a performance chart using a multi-tiered system of supports (MTS

Evaluate the following:
1. Accuracy
2. Specificity
3. MTSS Alignment (must be actionable, grounded in the chart, no reflection on past)
4. Clarity (≤300 words, ≤30 words/observation)

Insight:
{insight_text}

Please provide:
- Score (1–10) for each dimension
- Brief justification for each
- Suggested rewrites if necessary
If the insight is off-policy or missing, respond: 'No valid insight found.'
"""


# === STRUCTURED OUTPUT PARSING ===
# Extracts scores from the LLM's output response
# Looks for lines containing each rubric name and extracts the numeric score using regex

def parse_response(response_text):
    scores = {"Accuracy": None, "Specificity": None, "MTSS Alignment": None, "Clarity": None}
    for line in response_text.splitlines():
        for key in scores:
            if key in line:
                match = re.search(r'(\d+)', line)
                if match:
                    scores[key] = int(match.group(1))
    return scores


# === EVALUATION FUNCTION ===
# Evaluates a single chart and insight draft using Gemini 1.5 Pro
# Sends the image and text prompt to the model and returns its response

def evaluate_single(chart_path, insight_text):
    image = Image.open(chart_path)
    prompt = build_prompt(insight_text)
    response = model.generate_content([prompt, image])
    return response.text
```

```
# === BATCH RERANKING FUNCTION ===
# Evaluates multiple insight drafts per chart, scores each, and returns the best-scoring one
# Uses structured output parsing to compute average score and handles fail-safe detection

def rank_variants(chart_path, insight_variants):
    best_score = -1
    best_output = None
    best_variant = None
    for i, text in enumerate(insight_variants):
        print(f"\nEvaluating Variant {i+1}")
        raw_response = evaluate_single(chart_path, text)
        print(raw_response)
        if "No valid insight" in raw_response:
            continue  # Fail-safe: skip invalid or off-policy responses
        parsed_scores = parse_response(raw_response)
        if None not in parsed_scores.values():
            avg_score = sum(parsed_scores.values()) / 4
            if avg_score > best_score:
                best_score = avg_score
                best_output = raw_response
                best_variant = text
    return best_variant, best_output


# === CSV LOGGING ===
# Appends evaluation results (scores, raw insight, and LLM output) to a CSV file
# Automatically adds headers if the file does not exist yet

def log_to_csv(chart_name, variant_text, scores, full_response):
    header = ["chart", "accuracy", "specificity", "mtss_alignment", "clarity", "insight_text", "judge_output"]
    row = [chart_name, scores.get("Accuracy"), scores.get("Specificity"), scores.get("MTSS Alignment"),
           scores.get("Clarity"), variant_text, full_response]
    file_exists = os.path.exists(OUTPUT_CSV)
    with open(OUTPUT_CSV, 'a') as f:
        writer = csv.writer(f)
        if not file_exists:
            writer.writerow(header)
        writer.writerow(row)


# === MAIN EXECUTION LOOP ===
# Iterates over chart images, evaluates each set of insight variants,
# logs judge feedback to CSV, and saves the best insight to /best_insights/

best_insights_folder = "best_insights"
os.makedirs(best_insights_folder, exist_ok=True)

for filename in os.listdir(CHART_FOLDER):
    if not filename.endswith(".png"):
        continue

    chart_path = os.path.join(CHART_FOLDER, filename)
    base_name = os.path.splitext(filename)[0]
    insight_variants = []

    # Load all insight variants matching this chart
    for i in range(1, N_VARIANTS + 1):
        insight_path = os.path.join(INSIGHT_FOLDER, f"{base_name}_v{i}.txt")
        if os.path.exists(insight_path):
            with open(insight_path, "r") as f:
                insight_variants.append(f.read())

    if not insight_variants:
        print(f"No insight variants found for {base_name}")
        continue

    # Evaluate and rank variants using Judge LLM
    best_text, judge_output = rank_variants(chart_path, insight_variants)
    scores = parse_response(judge_output)

    if scores and best_text:
        # Log to CSV
        log_to_csv(base_name, best_text, scores, judge_output)

        # Export top insight to /best_insights/ as .txt
        output_path = os.path.join(best_insights_folder, f"{base_name}.txt")
        with open(output_path, "w") as f:
```

~~~
            f.write(best_text)
        print(f"✅ Saved top insight for {base_name} to {output_path}")


# === README.md ===
readme_text = """# 🧠 Judge LLM for Educational Insights (MTSS-Aligned)

This project uses Gemini 1.5 Pro to evaluate AI-generated school insights aligned to multi-tiered system of supports (MTSS). The

- Accuracy
- Specificity
- MTSS Alignment
- Clarity

It also:
- Selects the best variant among multiple insight drafts using average score
- Suggests rewrites if needed
- Filters out off-policy or incomplete insights

---

## 📂 Project Structure

```
/judge_llm_project/
├── charts/                # .png chart images
├── insights/              # .txt insight variants per chart
├── best_insights/         # top-ranked insights for dashboard use
├── judge_eval_results.csv # generated output
└── Judge Llm Eval.ipynb   # this notebook
```

Each chart must have multiple pre-generated insight variants:
```
chart1_v1.txt
chart1_v2.txt
chart1_v3.txt
```

These must be manually created or generated by a separate script or model *before* running this notebook. This notebook does not

If analyzing multiple schools separately, include the school name in the file prefix to group insights and charts together. Do *

### Example:

**Folder:** `/charts/`
```
schoolA_chart.png
schoolB_chart.png
```

**Folder:** `/insights/`
```
schoolA_chart_v1.txt
schoolA_chart_v2.txt
schoolA_chart_v3.txt

schoolB_chart_v1.txt
schoolB_chart_v2.txt
schoolB_chart_v3.txt
```

**Folder:** `/best_insights/`
```
schoolA_chart.txt   # highest-rated insight only
schoolB_chart.txt
```

---

## ✅ Insight Format

Each insight draft should follow this structure:

```
Blurb: A 1–2 sentence summary
~~~

~~~
Observations:
- Start with percentages or trends
- Avoid interpretation not visible in the chart
- Use ≤ 4 observations, each ≤ 30 words

Recommendations:
- School-wide support action
- Targeted action for at-risk groups
- Intensive intervention for high-need students
```

---

## 📊 Output

Logged to `judge_eval_results.csv`:
- Chart filename
- Scores for all 4 dimensions
- The original insight
- Full response from Judge LLM

Also saved to `/best_insights/` for dashboard integration:
- One `.txt` file per chart
- Contains only the top-rated insight body (no judge explanation)

---

## 🧠 Scoring Rubric

| Dimension       | Description |
|-----------------|-------------|
| Accuracy        | Are the observations grounded in the data? |
| Specificity     | Are numbers or metrics used instead of vague phrases? |
| MTSS Alignment  | Are recommendations immediately actionable, specific, and aligned to MTSS tiers? |
| Clarity         | Is the format readable, within word limits, and educator-friendly? |

---

## 🔁 Evaluation Behavior
- The judge scores all insight variants for a chart
- It chooses the highest-scoring insight using average score
- If an insight is off-policy, it returns: "No valid insight found."
- Results are logged to CSV
- Top insight text is written to `/best_insights/` as plain `.txt`

---

## 🛠 Extending the System

This notebook currently supports:
- Single-insight evaluation per image
- Reranking up to 3 variants using score average
- Score logging
- Top insight extraction for dashboard display
- Judge prompt for MTSS-aligned insight checking

Planned enhancements:
- Add a second prompt for Judge-generated rewrites of poor outputs
- Comparison prompt: let Judge pick between A vs B
- Prompt for MTSS-specific tier labeling suggestions

---

## 🚀 How to Use
1. Generate 2–3 insight variants per chart and save as `.txt` files in `/insights/`
2. Add matching `.png` chart files to `/charts/`
3. Prefix filenames with school names if analyzing multiple schools
4. Replace `YOUR_API_KEY` in the notebook
5. Run all cells to score, rank, and log the best insight
6. Read dashboard-ready outputs in `/best_insights/`

---

## 🔧 Future Extensions
- Reward model training (RLAIF)
- Self-revision using Judge feedback
- UI for batch testing or insight generation
~~~

```
"""

# Write README to file
with open("JudgeLLM_README.md", "w") as f:
    f.write(readme_text)
print("JudgeLLM_README.md created.")
```

```
JudgeLLM_README.md created.
```