AI PROJECT REPORT

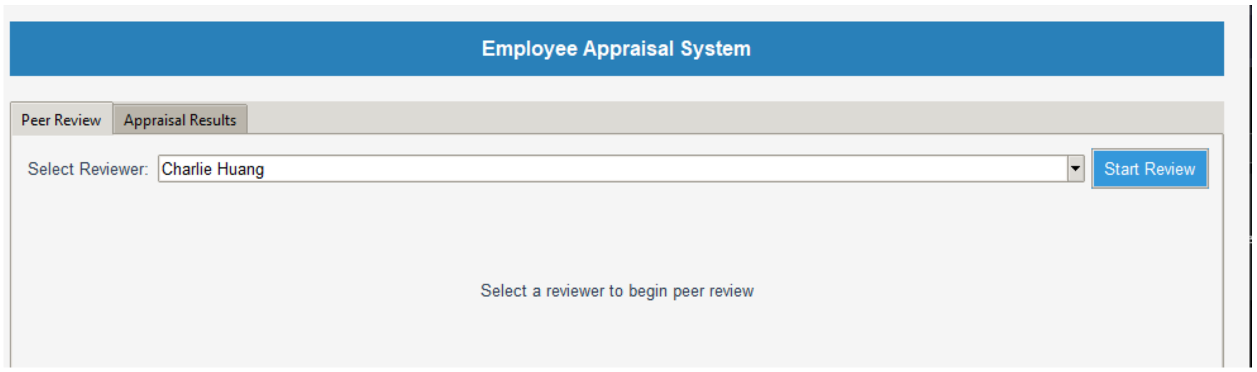
<u>Group Members:</u>

Hamza Naeem 22K-5063

Ali Hassan 22K-5010

Nafay Zaki 22K-5104

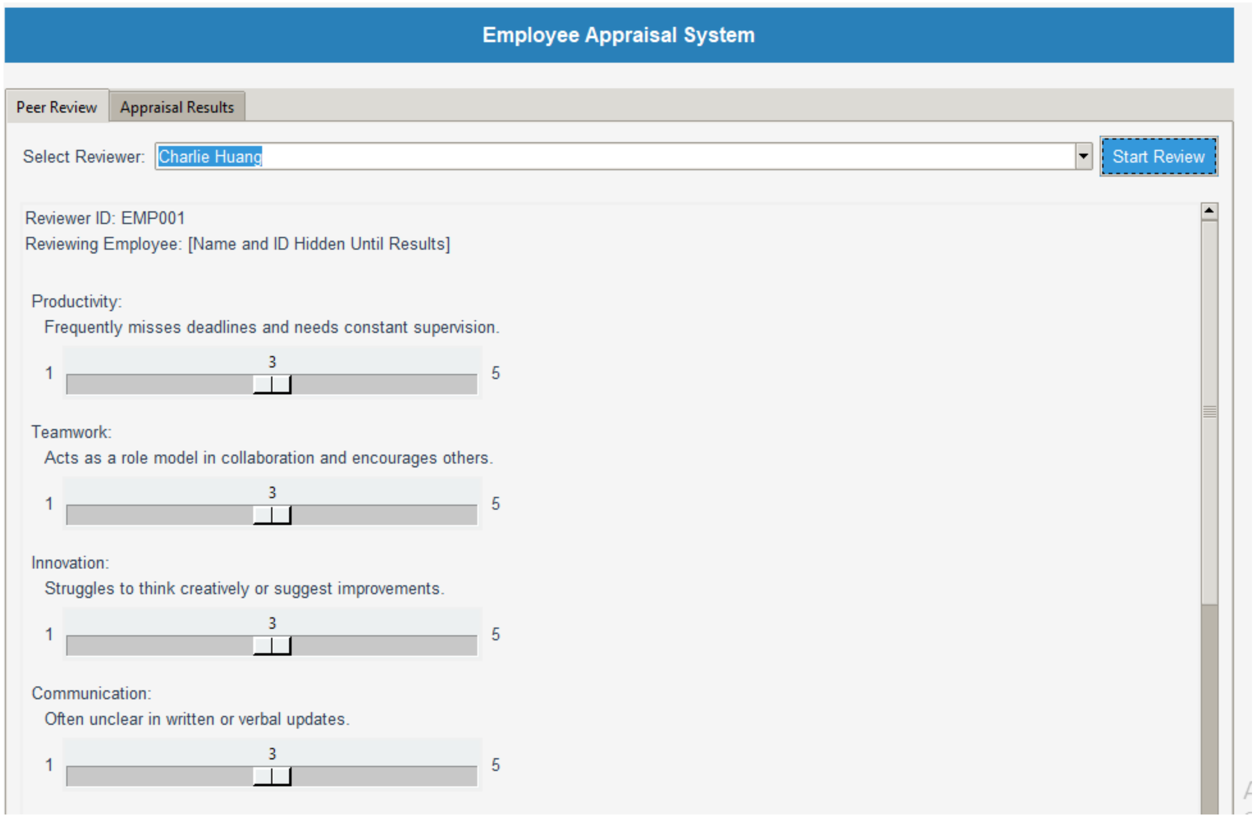
1. Initially the reviewer is supposed to select their name from the dropdown.



Select Reviewer: Charlie Huang

Select a reviewer to begin peer review

2. Proceed to the review process by initiating through “Start Review” click button.
3. Rate each category of the reviewee based on the objective details using the slide bar.


Employee Appraisal System

Select Reviewer: Charlie Huang

Reviewer ID: EMP001
Reviewing Employee: [Name and ID Hidden Until Results]

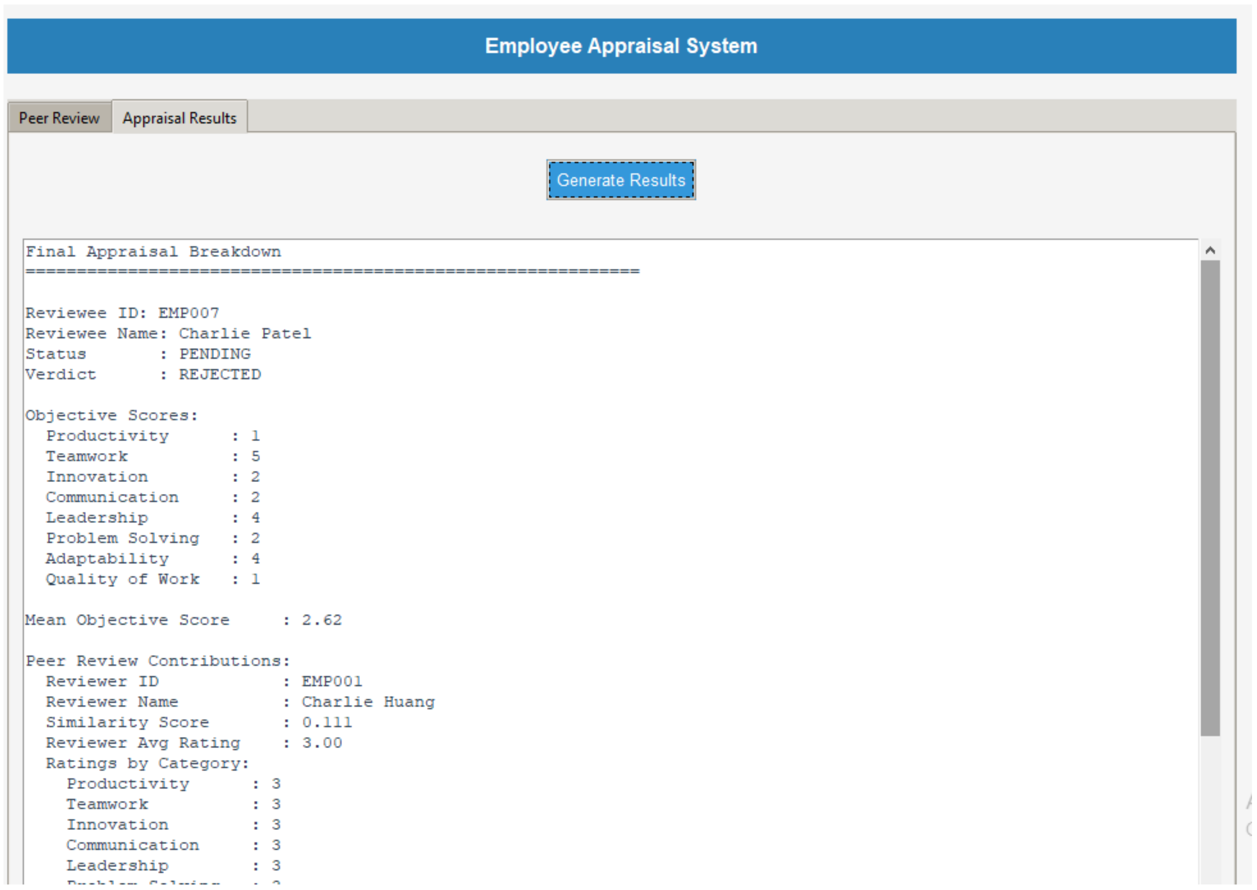
Productivity:
Frequently misses deadlines and needs constant supervision.
1 3 5

Teamwork:
Acts as a role model in collaboration and encourages others.
1 3 5

Innovation:
Struggles to think creatively or suggest improvements.
1 3 5

Communication:
Often unclear in written or verbal updates.
1 3 5

4. Submit the review using the “Submit Review” click button.
5. Generate appraisal results to view the breakdown and the related calculations.



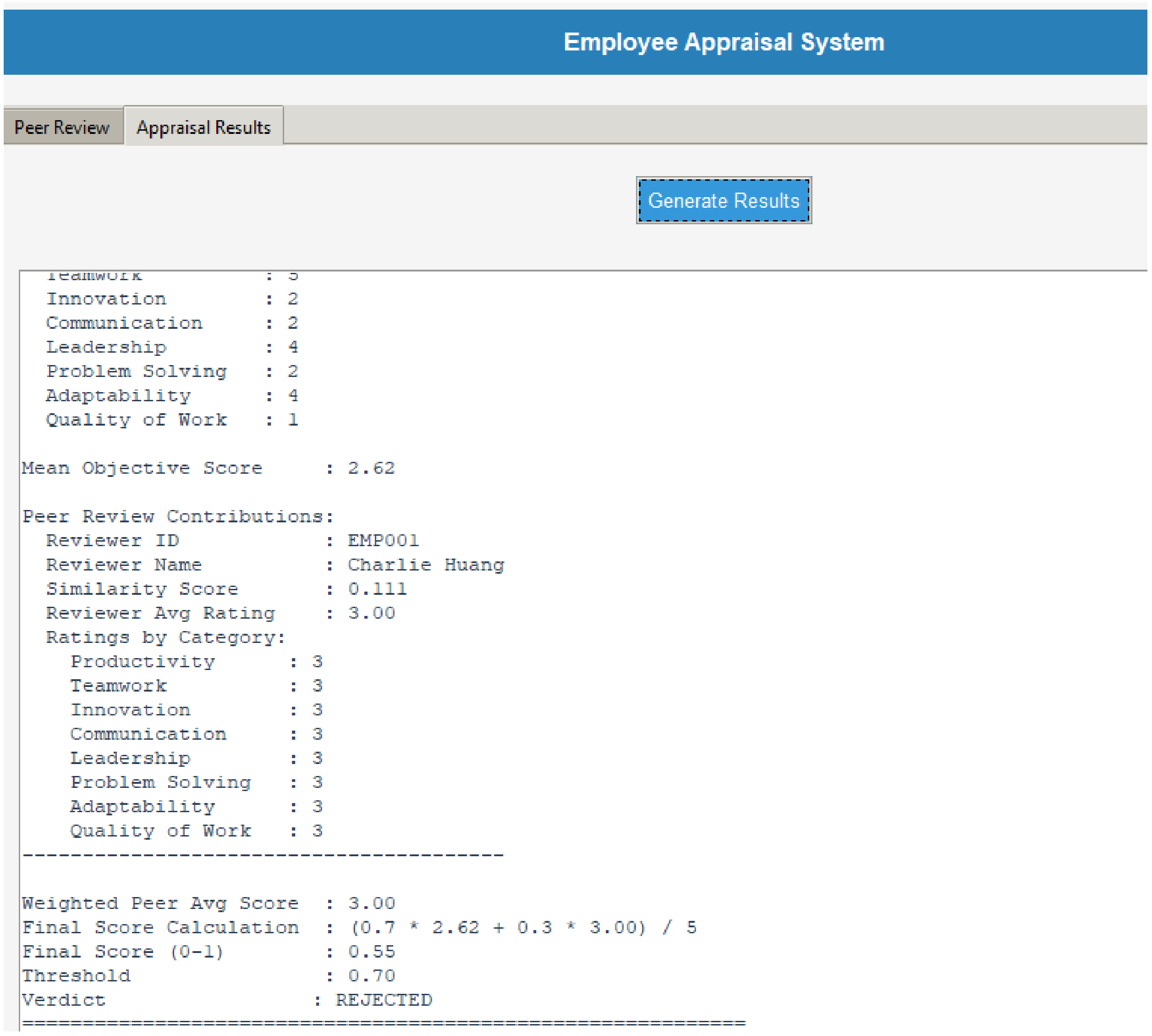
```
Final Appraisal Breakdown
============================================================

Reviewee ID: EMP007
Reviewee Name: Charlie Patel
Status       : PENDING
Verdict      : REJECTED

Objective Scores:
  Productivity      : 1
  Teamwork          : 5
  Innovation        : 2
  Communication     : 2
  Leadership        : 4
  Problem Solving   : 2
  Adaptability      : 4
  Quality of Work   : 1

Mean Objective Score     : 2.62

Peer Review Contributions:
  Reviewer ID            : EMP001
  Reviewer Name          : Charlie Huang
  Similarity Score       : 0.111
  Reviewer Avg Rating    : 3.00
  Ratings by Category:
    Productivity      : 3
    Teamwork          : 3
    Innovation        : 3
    Communication     : 3
    Leadership        : 3
```



```
  Teamwork          : 3
  Innovation        : 2
  Communication     : 2
  Leadership        : 4
  Problem Solving   : 2
  Adaptability      : 4
  Quality of Work   : 1

Mean Objective Score     : 2.62

Peer Review Contributions:
  Reviewer ID              : EMP001
  Reviewer Name            : Charlie Huang
  Similarity Score         : 0.111
  Reviewer Avg Rating      : 3.00
  Ratings by Category:
    Productivity      : 3
    Teamwork          : 3
    Innovation        : 3
    Communication     : 3
    Leadership        : 3
    Problem Solving   : 3
    Adaptability      : 3
    Quality of Work   : 3
----------------------------------------

Weighted Peer Avg Score  : 3.00
Final Score Calculation  : (0.7 * 2.62 + 0.3 * 3.00) / 5
Final Score (0-1)        : 0.55
Threshold                : 0.70
Verdict                 : REJECTED
============================================================
```

Overview of the Project:

This Employee Appraisal System is a comprehensive GUI application designed to streamline and automate the employee performance evaluation process. Built using Python's Tkinter for the interface, the system incorporates data analysis techniques to generate fair and insightful appraisal results. The project features employee management capabilities, allowing for the addition of employees with unique IDs and performance scores across key categories like Productivity, Teamwork, and Innovation. A core component is the blinded peer review process, where reviewers assess colleagues using slider-based ratings (1-5) while viewing objective performance descriptions rather than names, ensuring unbiased evaluations. The system employs sophisticated scoring mechanisms, combining 70% manager-assigned objective scores with 30% peer review ratings, weighted by cosine similarity to account for reviewer reliability when scikit-learn is available, with a fallback to simple averaging when it's not. The final appraisal results provide detailed performance breakdowns, including individual category scores, peer feedback with similarity metrics, and an overall APPROVED/REJECTED verdict based on a 0.7 threshold. The application demonstrates effective use of pandas for data management, NumPy for calculations, and randomization techniques for generating realistic demo data, all presented through an intuitive tabbed interface that guides users seamlessly from employee setup through review submission to final results analysis. This implementation successfully balances technical sophistication with user accessibility, offering organizations a practical tool for conducting fair, data-driven performance assessments.

Code:

```
import tkinter as tk
from tkinter import ttk, messagebox, scrolledtext
import random
import pandas as pd
```

```python
import numpy as np
from collections import defaultdict

# Handle sklearn import with fallback
try:
    from sklearn.metrics.pairwise import cosine_similarity
    sklearn_available = True
except ImportError:
    sklearn_available = False
    messagebox.showwarning("Warning", "scikit-learn not found. Using simplified
scoring.")

class EmployeeAppraisalSystem:
    def __init__(self, root):
        self.root = root
        self.root.title("Employee Appraisal System")
        self.root.geometry("1000x800")
        self.root.configure(bg="#f5f5f5")

        # Color scheme
        self.colors = {
            'primary': "#3498db",
            'secondary': "#2ecc71",
            'accent': "#e74c3c",
            'background': "#f5f5f5",
            'text': "#2c3e50",
            'header': "#2980b9",
            'panel': "#ecf0f1",
            'button': "#3498db",
            'button_text': "white"
        }

        self.categories = [
            'Productivity', 'Teamwork', 'Innovation', 'Communication',
            'Leadership', 'Problem Solving', 'Adaptability', 'Quality of Work'
        ]
        self.employees = []
        self.employee_codes = {}
        self.objective_details = pd.DataFrame()
        self.reviews = defaultdict(list)
        self.reviewed_pairs = set()

        self.configure_styles()
        self.generate_demo_employees(10)  # Generate demo data first
        self.create_main_interface()      # Then create interface

    def configure_styles(self):
        style = ttk.Style()
        style.theme_use('clam')

        # Frame styles
        style.configure('TFrame', background=self.colors['background'])
        style.configure('Header.TFrame', background=self.colors['header'])

        # Label styles
        style.configure('TLabel',
                      background=self.colors['background'],
                      foreground=self.colors['text'],
                      font=('Helvetica', 10))
        style.configure('Header.TLabel',
                      background=self.colors['header'],
                      foreground='white',
                      font=('Helvetica', 12, 'bold'))

        # Button styles
        style.configure('TButton',
                      background=self.colors['button'],
                      foreground=self.colors['button_text'],
                      font=('Helvetica', 10))

        # Combobox styles
        style.map('TCombobox',
                 fieldbackground=[('readonly', 'white')],
                 selectbackground=[('readonly', self.colors['primary'])])

    def generate_demo_employees(self, num):
        first_names = ['Alice', 'Bob', 'Charlie', 'Diana', 'Eve', 'Frank']
        last_names = ['Nguyen', 'Patel', 'Rodriguez', 'Kim', 'Huang', 'Lee']

        for i in range(num):
            name = f"{random.choice(first_names)} {random.choice(last_names)}"
            emp_id = f"EMP{i+1:03d}"
            self.employees.append(name)
            self.employee_codes[name] = emp_id

            scores = {cat: random.randint(1, 5) for cat in self.categories}
            if self.objective_details.empty:
                self.objective_details = pd.DataFrame(scores, index=[name])
            else:
                self.objective_details.loc[name] = scores

    def create_main_interface(self):
        # Main container
        main_frame = ttk.Frame(self.root)
        main_frame.pack(fill=tk.BOTH, expand=True, padx=20, pady=20)

        # Header
        header_frame = ttk.Frame(main_frame, style='Header.TFrame')
        header_frame.pack(fill=tk.X, pady=(0, 20))

        ttk.Label(header_frame,
                 text="Employee Appraisal System",
                 style='Header.TLabel').pack(pady=10)

        # Tab control
        self.tab_control = ttk.Notebook(main_frame)

        # Create tabs
        self.create_review_tab()
        self.create_results_tab()

        self.tab_control.pack(fill=tk.BOTH, expand=True)

    def create_review_tab(self):
        # Review tab
        review_tab = ttk.Frame(self.tab_control)
        self.tab_control.add(review_tab, text="Peer Review")

        # Review controls frame
        controls_frame = ttk.Frame(review_tab)
        controls_frame.pack(fill=tk.X, padx=10, pady=10)

        ttk.Label(controls_frame, text="Select Reviewer:").pack(side=tk.LEFT)

        # Create combobox with proper configuration
        self.reviewer_combo = ttk.Combobox(
            controls_frame,
            values=self.employees,
            state="readonly",
            height=10,
            font=('Helvetica', 10)
        )
        self.reviewer_combo.pack(side=tk.LEFT, padx=5, fill=tk.X, expand=True)

        # Set default value if employees exist
        if self.employees:
            self.reviewer_combo.current(0)

        ttk.Button(controls_frame,
                   text="Start Review",
                   command=self.start_review).pack(side=tk.LEFT)

        # Review display area
        self.review_display = ttk.Frame(review_tab)
        self.review_display.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        # Initial message
        ttk.Label(self.review_display,
                  text="Select a reviewer to begin peer review").pack(pady=50)

    def create_results_tab(self):
        # Results tab
        results_tab = ttk.Frame(self.tab_control)
        self.tab_control.add(results_tab, text="Appraisal Results")

        # Controls frame
        controls_frame = ttk.Frame(results_tab)
        controls_frame.pack(fill=tk.X, padx=10, pady=10)

        ttk.Button(controls_frame,
                   text="Generate Results",
                   command=self.display_results).pack(pady=10)

        # Results display
        self.results_text = scrolledtext.ScrolledText(
            results_tab,
            wrap=tk.WORD,
            width=100,
            height=30,
            font=('Courier', 10),
            bg='white',
            fg=self.colors['text']
        )
        self.results_text.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

    def start_review(self):
        reviewer = self.reviewer_combo.get()

        # Validate selection
        if not reviewer or reviewer not in self.employees:
            messagebox.showerror("Error", "Please select a valid reviewer from
the dropdown list")
            return

        available = [e for e in self.employees if e != reviewer and (reviewer, e)
not in self.reviewed_pairs]
        if not available:
            messagebox.showinfo("Info", "No more employees available for this
reviewer to review")
            return

        self.current_reviewee = random.choice(available)
        self.current_reviewer = reviewer

        # Clear previous review display
        for widget in self.review_display.winfo_children():
            widget.destroy()

        self.setup_review_interface()

    def setup_review_interface(self):
        # Create canvas and scrollbar
        canvas = tk.Canvas(self.review_display, bg=self.colors['background'])
        scrollbar = ttk.Scrollbar(self.review_display, orient="vertical",
command=canvas.yview)
        scrollable_frame = ttk.Frame(canvas)

        # Configure scrollregion
        scrollable_frame.bind(
            "<Configure>",
            lambda e: canvas.configure(scrollregion=canvas.bbox("all")))

        canvas.create_window((0, 0), window=scrollable_frame, anchor="nw")
        canvas.configure(yscrollcommand=scrollbar.set)
        canvas.pack(side="left", fill="both", expand=True)
        scrollbar.pack(side="right", fill="y")

        # Add content to scrollable frame
        ttk.Label(scrollable_frame,
                 text=f"Reviewer ID:
{self.employee_codes[self.current_reviewer]}").pack(anchor="w")
        ttk.Label(scrollable_frame,
                 text="Reviewing Employee: [Name and ID Hidden Until
Results]").pack(anchor="w", pady=(0, 20))

        scores = self.objective_details.loc[self.current_reviewee]
        self.rating_vars = {}

        for cat in self.categories:
            frame = ttk.Frame(scrollable_frame)
            frame.pack(fill=tk.X, padx=5, pady=5)

            ttk.Label(frame, text=f"{cat}:").pack(anchor="w")
            desc = self._get_description(scores[cat], cat)
            ttk.Label(frame, text=desc, wraplength=700).pack(anchor="w", padx=10)

            # Rating slider
            slider_frame = ttk.Frame(frame)
            slider_frame.pack(fill=tk.X, padx=10, pady=5)

            ttk.Label(slider_frame, text="1").pack(side="left")
            var = tk.IntVar(value=3)
            tk.Scale(slider_frame, from_=1, to=5, orient="horizontal",
                   variable=var, bg=self.colors['panel']).pack(side="left",
padx=5, fill=tk.X, expand=True)
            ttk.Label(slider_frame, text="5").pack(side="left")

            self.rating_vars[cat] = var

        # Submit button
        ttk.Button(scrollable_frame,
                  text="Submit Review",
                  command=self.submit_review).pack(pady=20)

    def submit_review(self):
        ratings = {cat: var.get() for cat, var in self.rating_vars.items()}
        self.reviews[self.current_reviewee].append((self.current_reviewer,
ratings))
        self.reviewed_pairs.add((self.current_reviewer, self.current_reviewee))

        for widget in self.review_display.winfo_children():
            widget.destroy()

        ttk.Label(self.review_display, text="Review submitted
successfully!").pack(pady=50)

        # Refresh combobox
        self.reviewer_combo['values'] = self.employees
        if self.employees:
            self.reviewer_combo.current(0)

    def display_results(self):
        self.results_text.delete(1.0, tk.END)
        scores = self.analyze_reviews()

        if not scores:
            self.results_text.insert(tk.END, "No appraisal data available yet.")
            return

        self.results_text.insert(tk.END, "Final Appraisal Breakdown\n")
        self.results_text.insert(tk.END, "="*60 + "\n")

        for emp, details in scores.items():
            self.format_employee_results(emp, details)

    def format_employee_results(self, emp, details):
        self.results_text.insert(tk.END, f"\nReviewee ID:
{self.employee_codes[emp]}\n")
        self.results_text.insert(tk.END, f"Reviewee Name: {emp}\n")
        self.results_text.insert(tk.END, f"Status       : {details['status']}\n")
        self.results_text.insert(tk.END, f"Verdict      :
{details['verdict']}\n\n")

        self.results_text.insert(tk.END, "Objective Scores:\n")
        for cat, score in details['objective_scores'].items():
            self.results_text.insert(tk.END, f"  {cat:<18}: {score}\n")

        self.results_text.insert(tk.END, f"\nMean Objective Score     :
{details['objective_mean']:.2f}\n\n")
```

```python
        self.results_text.insert(tk.END, "Peer Review Contributions:\n")

        for review in details['reviewer_details']:
            self.format_review_details(review)

        self.results_text.insert(tk.END, f"\nWeighted Peer Avg Score   : 
{details['weighted_peer_avg']:.2f}\n")
        self.results_text.insert(tk.END, f"Final Score Calculation   : (0.7 * 
{details['objective_mean']:.2f} + 0.3 * {details['weighted_peer_avg']:.2f}) / 
5\n")
        self.results_text.insert(tk.END, f"Final Score (0-1)         : 
{details['final_score']:.2f}\n")
        self.results_text.insert(tk.END, f"Threshold                 : 0.70\n")
        self.results_text.insert(tk.END, f"Verdict                  : 
{details['verdict']}\n")
        self.results_text.insert(tk.END, "="*60 + "\n")

    def format_review_details(self, review):
        reviewer, ratings, sim, avg_rating = review
        self.results_text.insert(tk.END, f"  Reviewer ID             : 
{self.employee_codes[reviewer]}\n")
        self.results_text.insert(tk.END, f"  Reviewer Name           : 
{reviewer}\n")
        self.results_text.insert(tk.END, f"  Similarity Score        : 
{sim:.3f}\n")
        self.results_text.insert(tk.END, f"  Reviewer Avg Rating     : 
{avg_rating:.2f}\n")
        self.results_text.insert(tk.END, "  Ratings by Category:\n")

        for cat, rating in ratings.items():
            self.results_text.insert(tk.END, f"    {cat:<18}: {rating}\n")

        self.results_text.insert(tk.END, "-"*40 + "\n")

    def analyze_reviews(self):
        if not self.reviews:
            return {}

        if sklearn_available:
            return self.analyze_with_sklearn()
        else:
            return self.analyze_without_sklearn()
    def analyze_with_sklearn(self):
        matrix = self.build_user_item_matrix()
        cosine_sim = cosine_similarity(matrix)
        sim_df = pd.DataFrame(cosine_sim, index=self.employees,
columns=self.employees)

        final_scores = {}
        for emp in self.reviews.keys():
            final_scores[emp] = self.calculate_employee_score(emp, sim_df)

        return final_scores

    def analyze_without_sklearn(self):
        final_scores = {}
        for emp in self.reviews.keys():
            final_scores[emp] = self.simple_calculate_employee_score(emp)

        return final_scores

    def build_user_item_matrix(self):
        matrix = pd.DataFrame(index=self.employees, columns=self.employees,
dtype=float)
        for reviewee in self.reviews:
            for reviewer, rating in self.reviews[reviewee]:
                avg_score = np.mean(list(rating.values()))
                matrix.loc[reviewer, reviewee] = avg_score
        return matrix.fillna(0)

    def calculate_employee_score(self, emp, sim_df):
        obj_scores = self.objective_details.loc[emp]
        obj_mean = obj_scores.mean()
        peer_reviews = self.reviews.get(emp, [])

        weighted_peer_sum = 0.0
        total_weight = 0.0
        reviewer_details = []

        for reviewer, rating in peer_reviews:
            avg_rating = np.mean(list(rating.values()))
            similarity_row = sim_df.loc[reviewer]
            similarity = similarity_row.drop(emp).mean() if emp in similarity_row
else similarity_row.mean()
```

```
            weighted_peer_sum += avg_rating * similarity
            total_weight += similarity
            reviewer_details.append((reviewer, rating, similarity, avg_rating))

        weighted_peer_avg = weighted_peer_sum / total_weight if total_weight > 0
else 0.0
        final_score = (0.7 * obj_mean + 0.3 * weighted_peer_avg) / 5.0

        status = "COMPLETE" if len(peer_reviews) >= 3 else "PENDING"
        verdict = "APPROVED" if final_score >= 0.7 else "REJECTED"

        return {
            'final_score': final_score,
            'objective_scores': obj_scores.to_dict(),
            'objective_mean': obj_mean,
            'reviewer_details': reviewer_details,
            'weighted_peer_avg': weighted_peer_avg,
            'status': status,
            'verdict': verdict
        }

    def simple_calculate_employee_score(self, emp):
        obj_scores = self.objective_details.loc[emp]
        obj_mean = obj_scores.mean()
        peer_reviews = self.reviews.get(emp, [])

        peer_avg = np.mean([np.mean(list(rating.values())) for _, rating in
peer_reviews]) if peer_reviews else 0.0
        final_score = (0.7 * obj_mean + 0.3 * peer_avg) / 5.0

        status = "COMPLETE" if len(peer_reviews) >= 3 else "PENDING"
        verdict = "APPROVED" if final_score >= 0.7 else "REJECTED"

        reviewer_details = [
            (reviewer, rating, 0.0, np.mean(list(rating.values())))
            for reviewer, rating in peer_reviews
        ]

        return {
            'final_score': final_score,
            'objective_scores': obj_scores.to_dict(),
            'objective_mean': obj_mean,
            'reviewer_details': reviewer_details,
```

```python
            'weighted_peer_avg': peer_avg,
            'status': status,
            'verdict': verdict
        }

    def _get_description(self, score, category):
        descriptions = {
            'Productivity': [
                "Frequently misses deadlines and needs constant supervision.",
                "Delivers some tasks but misses project timelines often.",
                "Meets basic project expectations and completes work on time.",
                "Proactively completes tasks, often ahead of schedule.",
                "Delivers exceptional work with efficiency and self-discipline."
            ],
            'Teamwork': [
                "Rarely collaborates and struggles in team environments.",
                "Inconsistent in collaboration or communicating ideas.",
                "Participates in team discussions and shares responsibilities.",
                "Supports teammates proactively and shares credit.",
                "Acts as a role model in collaboration and encourages others."
            ],
            'Innovation': [
                "Lacks new ideas and sticks to routine solutions.",
                "Struggles to think creatively or suggest improvements.",
                "Contributes some practical ideas occasionally.",
                "Proposes relevant optimizations and automation opportunities.",
                "Leads innovation with new feature ideas and process changes."
            ],
            'Communication': [
                "Struggles to convey ideas and causes confusion in teams.",
                "Often unclear in written or verbal updates.",
                "Communicates adequately with clients and teammates.",
                "Explains ideas clearly and keeps stakeholders informed.",
                "Ensures clarity in meetings, emails, and client interactions."
            ],
            'Leadership': [
                "Avoids taking initiative or leading discussions.",
                "Rarely steps up or volunteers for responsibilities.",
                "Manages tasks when assigned and delegates decently.",
                "Guides peers during sprints and provides mentorship.",
                "Inspires others and manages complex situations gracefully."
            ],
            'Problem Solving': [
                "Needs support to resolve even minor bugs or blockers.",
                "Takes too long to identify or resolve issues.",
                "Can independently resolve moderate technical problems.",
                "Efficient in debugging and architectural analysis.",
                "Resolves issues others struggle with, often preventing them 
early."
            ],
            'Adaptability': [
                "Struggles to adjust to changing project requirements.",
                "Takes longer to adapt to new tools or workflows.",
                "Adapts decently under deadline pressures.",
                "Quickly adopts tech stacks and agile changes.",
                "Thrives in changing environments and adopts new practices 
rapidly."
            ],
            'Quality of Work': [
                "Code or documentation often lacks testing and clarity.",
                "Makes errors requiring frequent reviews or rollbacks.",
                "Produces work that passes reviews with minor revisions.",
                "Delivers maintainable, well-tested modules.",
                "Delivers polished, scalable, and reliable software 
consistently."
            ]
        }
        return descriptions[category][score - 1]

if __name__ == "__main__":
    root = tk.Tk()
    app = EmployeeAppraisalSystem(root)
    root.mainloop()
```