Author: Madhurima Rawat

Agile Sprint Planning

Agile Sprint Planning: Conduct a simulated sprint planning session following agile principles, where students break down user stories into tasks, estimate their effort, and allocate them for a sprint.

1. Project Overview: Agile Sprint Planning

Goal:

- Conduct a simulated sprint planning session using Agile principles.
- Break down user stories into tasks, estimate their effort, and allocate them among team members (Alice, Bob, Charlie).
- Visualize and analyze sprint data using Streamlit.

2. Example Agile Sprint Planning Workflow

User Stories Example:

Dataset and sprint task list tailored specifically to the described project, focusing on tasks like data collection, pipeline creation, deployment, and bug fixing.

Dataset Example for Agile Sprint Planning

Story ID	Description	Priority	Effort	Assigned To
1	Collect dataset and clean for processing.	Medium	4	Alice
2	Create a pipeline for visualizing sprint data.	High	6	Bob
3	Deploy Streamlit app to cloud hosting.	High	8	Charlie
4	Fix app crash caused by missing input validation.	High	9	Alice
5	Implement effort estimation slider for tasks.	Medium	5	Bob
6	Add team assignment logic for user stories.	Low	3	Charlie

Steps for Task Breakdown and Assignment

1. **Task Identification**: Break the project into actionable tasks (e.g., collect data, fix bugs, deploy app).
2. **Effort Estimation**: Use points to estimate the effort needed for each task (1 = low, 10 = high).
3. **Prioritization**: Assign priority levels (High, Medium, Low) based on project needs.
4. **Team Assignment**: Allocate tasks evenly among team members to balance workload.

Generated Sprint Plan

Tasks with Points, Priorities, and Assignments

Task	Points (Effort)	Priority	Assigned To
Data collection and preprocessing	4	Medium	Alice
Build sprint visualization pipeline	6	High	Bob
Deploy app to Streamlit cloud	8	High	Charlie
Fix crash on invalid user input	9	High	Alice
Add effort slider for estimation	5	Medium	Bob
Add team assignment functionality	3	Low	Charlie

Sprint Summary

- **Total Effort**: 35 points.
- **Average Effort per Team Member**:
  - Alice: 13 points.
  - Bob: 11 points.
  - Charlie: 11 points.

Effort Distribution Visualization

- A bar chart showing points distributed across Alice, Bob, and Charlie.
- Ensures balance and highlights potential overloads.

This setup uses Agile principles to organize, estimate, and allocate tasks efficiently, making the project manageable and transparent.

3. Simple Python Script to Analyze Iris Dataset

Code:

```
# Import libraries
import pandas as pd
import seaborn as sns

# Load the Iris dataset
iris = sns.load_dataset("iris")

# Print dataset overview
print("Iris Dataset Overview:")
print(iris.head())  # Shows first few rows of the dataset

# Print data types and number of missing values
print("\nData Types and Missing Values:")
print(iris.info())

# Summary statistics
print("\nSummary Statistics:")
print(iris.describe())  # Provides count, mean, std, min, max, and quantiles

# Count of unique values in each column
print("\nCount of unique values in each column:")
print(iris.nunique())

# Class distribution (Count of each species)
print("\nClass Distribution (Species counts):")
print(iris["species"].value_counts())

# Display additional statistics (e.g., mean, std, min, max)
print("\nDetailed Statistics (including mean, std, min, max for each feature):")
print(iris.describe(include="all"))
```

Console Output:

1. **Dataset Overview**: First few rows of the Iris dataset.
2. **Summary Statistics**: Min, max, mean, and other descriptive stats for each feature.
3. **Graph**: Pair plot with species-wise grouping.

4. Code for Agile Sprint Planning Simulation

Integration of the Agile Sprint Planning and dataset visualization in Streamilt:

Streamlit Implementation:

```python
# Importing required libraries
import streamlit as st
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt


# Function to include background image and opacity
def display_background_image(url, opacity):
    """
    Displays a background image with a specified opacity on the web app using CSS.

    Args:
    - url (str): URL of the background image.
    - opacity (float): Opacity level of the background image.
    """
    # Set background image using HTML and CSS
    st.markdown(
        f"""
        <style>
            body {{
                background: url('{url}') no-repeat center center fixed;
                background-size: cover;
                opacity: {opacity};
            }}
        </style>
        """,
        unsafe_allow_html=True,
    )


# Call function to display the background image with opacity
display_background_image(
    "https://1.bp.blogspot.com/-NFhT7E4-TUQ/XqWXsuNJmJI/AAAAAAAALZw/iR8sFnKfwz89GD1ihOlNujDtLr
    0.8,
)


# App Title
st.title("Agile Sprint Planning & Data Analysis")
st.subheader("Simulate sprint planning and analyze datasets using Agile principles.")

# Data for Agile Sprint Planning
data = {
    "Task": [
        "Data collection and preprocessing",
        "Build sprint visualization pipeline",
        "Deploy app to Streamlit cloud",
        "Fix crash on invalid user input",
        "Add effort slider for estimation",
        "Add team assignment functionality",
    ],
    "Points (Effort)": [4, 6, 8, 9, 5, 3],
    "Priority": ["Medium", "High", "High", "High", "Medium", "Low"],
    "Assigned To": ["Alice", "Bob", "Charlie", "Alice", "Bob", "Charlie"],
}

df = pd.DataFrame(data)

# Display Task Table
st.header("Sprint Tasks Overview")
st.dataframe(df)

# Effort and Priority Distribution for Each Team Member
st.subheader("Effort and Priority Distribution for Each Team Member")
team_members = df["Assigned To"].unique()

# Colors for each chart with a valid pastel color palette
effort_colors = sns.color_palette("pastel", len(df["Assigned To"].unique()))
priority_colors = sns.color_palette("viridis", len(df["Priority"].unique()))

# Adjust the size of the pie charts
for member in team_members:
    member_tasks = df[df["Assigned To"] == member]

    # Effort Distribution Pie Chart
    fig_effort, ax_effort = plt.subplots(figsize=(15, 15))  # Same size for uniformity
    member_tasks.plot.pie(
        y="Points (Effort)",
        labels=member_tasks["Task"],
        autopct="%1.1f%%",
        colors=effort_colors,  # Different pastel colors for effort chart
        ax=ax_effort,
        legend=False,
        startangle=90,
        textprops={"fontsize": 8, "fontweight": "bold"},  # Make percentage values bold
    )
    ax_effort.set_title(
        f"Effort Distribution: {member}", fontsize=35, fontweight="bold"
    )
    ax_effort.set_ylabel("")  # Remove default y-axis label
    ax_effort.set_aspect("equal")  # Ensure the pie chart is circular
    ax_effort.legend(fontsize=10, prop={"weight": "bold"})  # Make the legend bold

    # Priority Distribution Pie Chart
    priority_counts = member_tasks["Priority"].value_counts()
    fig_priority, ax_priority = plt.subplots(figsize=(5, 5))  # Same size for uniformity
    priority_counts.plot.pie(
        autopct="%1.1f%%",
        colors=priority_colors,  # Different pastel colors for priority chart
        ax=ax_priority,
        startangle=90,
        textprops={"fontsize": 8, "fontweight": "bold"},  # Make percentage values bold
    )
    ax_priority.set_title(
        f"Priority Distribution: {member}", fontsize=12, fontweight="bold"
    )
    ax_priority.set_ylabel("")  # Remove default y-axis label
    ax_priority.set_aspect("equal")  # Ensure the pie chart is circular
    ax_priority.legend(fontsize=10, prop={"weight": "bold"})  # Make the legend bold

    # Display both charts side by side in columns
    col1, col2 = st.columns(2)
    with col1:
        st.pyplot(fig_effort, use_container_width=True)
    with col2:
        st.pyplot(fig_priority, use_container_width=True)


# Task Complexity Distribution
st.subheader("Task Complexity Distribution (Priority Levels)")
priority_distribution = df["Priority"].value_counts()
fig, ax = plt.subplots(figsize=(6, 6))
priority_distribution.plot.pie(
    autopct="%1.1f%%",
    colors=sns.color_palette("pastel"),
    ax=ax,
    startangle=90,
    textprops={"fontsize": 10, "fontweight": "bold"},
)
ax.set_title("Task Complexity Distribution by Priority", fontsize=12, fontweight="bold")
ax.set_ylabel("")  # Remove default y-axis label
st.pyplot(fig)

# Load Iris Dataset for Comparison
st.header("Data Analysis: Iris Dataset")
if st.checkbox("Analyze Iris Dataset"):
    # Load dataset
    iris = sns.load_dataset("iris")
    st.write("Iris Dataset Overview:")
    st.dataframe(iris.head())

    # Species Distribution Pie Chart
    st.subheader("Species Distribution")
    species_distribution = iris["species"].value_counts()
    fig, ax = plt.subplots(figsize=(6, 6))
    species_distribution.plot.pie(
        autopct="%1.1f%%",
        colors=sns.color_palette("pastel"),
        ax=ax,
        startangle=90,
        textprops={"fontsize": 10, "fontweight": "bold"},
    )
    ax.set_title("Species Distribution", fontweight="bold", fontsize=12)
    plt.xticks(fontweight="bold")
    plt.yticks(fontweight="bold")
    plt.xlabel("Species", fontweight="bold")
    ax.set_ylabel("", fontweight="bold")  # Remove default y-axis label
```

```python
st.pyplot(fig)

# Scatter Plot
st.subheader("Scatter Plot: Sepal Dimensions")
fig, ax = plt.subplots(figsize=(8, 6))
sns.scatterplot(
    x="sepal_length",
    y="sepal_width",
    hue="species",
    data=iris,
    palette="pastel",
    alpha=0.8,
    ax=ax,
)
ax.set_title("Sepal Length vs Sepal Width by Species", fontweight="bold")
plt.xticks(fontweight="bold")
plt.yticks(fontweight="bold")
plt.xlabel("Sepal Length", fontweight="bold")
plt.ylabel("Sepal Width", fontweight="bold")
# Set legend with bold fontweight
plt.legend(loc="upper right", prop={"weight": "bold"})
st.pyplot(fig)

# Box Plot
st.subheader("Box Plot: Petal Dimensions")
fig, ax = plt.subplots(figsize=(8, 6))
sns.boxplot(x="species", y="petal_length", data=iris, palette="pastel", ax=ax)
plt.xticks(fontweight="bold")
plt.yticks(fontweight="bold")
plt.xlabel("Species", fontweight="bold")
plt.ylabel("Petal Length", fontweight="bold")
ax.set_title("Petal Length Distribution by Species", fontweight="bold")
st.pyplot(fig)

# Histogram
st.subheader("Histogram: Petal Width Distribution")
fig, ax = plt.subplots(figsize=(8, 6))
sns.histplot(
    iris,
    x="petal_width",
    hue="species",
    kde=True,
    palette="pastel",
    alpha=0.8,
    ax=ax,
)
ax.set_title("Petal Width Distribution by Species", fontweight="bold")
plt.xticks(fontweight="bold")
plt.yticks(fontweight="bold")
plt.xlabel("Species", fontweight="bold")
plt.ylabel("Petal Length", fontweight="bold")
st.pyplot(fig)
```

```
    # Pairplot for Iris Dataset
    st.subheader("Pairplot: Sepal and Petal Dimensions")
    pair_plot = sns.pairplot(iris, hue="species", palette="pastel")
    st.pyplot(pair_plot)

# Deployment Notes
st.header("Deployment Steps")
st.markdown(
    """
### **1. Create GitHub Repository**
- Push all code (including this Streamlit app) to a repository.

### **2. Add `requirements.txt`**
Create a file named `requirements.txt` in your repository with the following content:
`streamlit pandas matplotlib seaborn`

### **3. Deploy to Streamlit Cloud**
- Log in to [Streamlit Cloud](https://streamlit.io/).
- Connect your GitHub repository.
- Deploy your app directly by selecting the `main` branch.

### **4. Test the App**
- Open the deployed app link.
- Test file upload, effort estimation, task allocation, and visualizations.
"""
)

# Footer with Author Info
st.write(
    """
**Analyze Effort Distribution and Task Allocation to improve sprint outcomes.**
---
**Author**: Madhurima Rawat | © 2024
"""
)
```

5. Explanation of Agile Sprint Process

1. **Effort Estimation:**
   - Use the **slider** widget to estimate effort for each user story (range: 1–10).
   - Effort is based on complexity and priority.
2. **Task Allocation:**
   - Assign stories to team members using the **dropdown menu**.
   - The system ensures that tasks are distributed evenly among Alice, Bob, and Charlie.
3. **Visualization:**

- **Bar chart** showing effort distribution among team members.
- Helps identify if workloads are balanced.

Summary Table

Step	Code/Action	Output
Console Analysis	Simple Python script to analyze Iris dataset with `pandas` and visualize with `seaborn`.	Dataset overview, summary statistics, pair plot
Streamlit Integration	Modified script to display dataset, stats, and plot interactively in Streamlit.	Interactive Streamlit app
Deployment	Set up GitHub repo, add `requirements.txt`, deploy to Streamlit Cloud.	Deployed app link

6. Deployment Pipeline

Steps:

1. **Set Up GitHub Repository:**
   - Push code and sample datasets to GitHub.
2. **Create `requirements.txt`:**

```
streamlit
pandas
matplotlib
```

3. **Deploy on Streamlit Cloud:**
   - Log in to Streamlit Cloud.
   - Link GitHub repository and deploy the app.

7. Evaluation

Pros:

- **Realistic Simulation:** Provides an intuitive understanding of Agile principles through a real-world example, using the Iris dataset to simulate sprint planning.

- **Interactive UI:** User-friendly and easy to navigate, allowing seamless interaction with the data and visualizations.

Cons:

- **Limited Scalability:** The tool is designed for simpler datasets, making it challenging to scale for larger projects or more complex datasets that require advanced planning.
- **Static Assignments:** Effort distribution is fixed, lacking dynamic adjustment based on changing project conditions or team availability.

Future Enhancements:

- Implement AI/ML models to automate effort estimation for more accurate planning.
- Add drag-and-drop Kanban boards for enhanced task visualization and improved sprint management.