```
# Import necessary libraries
import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

# Streamlit app configuration
st.set_page_config(
    page_title="Middle School Data Dashboard",
    layout="wide",
    initial_sidebar_state="expanded"
)

# Title and header
st.title("Middle School Data Dashboard")
st.header("Analyze and Visualize Student Performance")

# Sidebar for user input
st.sidebar.header("Upload Your Data")
uploaded_file = st.sidebar.file_uploader(
    "Upload a CSV file with student data", type=["csv"]
)

# Required columns
required_columns = [
    'Last Name', 'First Name', 'Student ID', 'Student Grade',
'Academic Year',
    'School', 'Subject', 'Enrolled', 'Sex', 'Hispanic or Latino',
'Race',
    'English Language Learner', 'Special Education', 'Economically
Disadvantaged',
    'Migrant', 'Class(es)', 'Class Teacher(s)', 'Report Group(s)',
'Date Range',
    'Date Range Start', 'Date Range End', 'Total Lesson Time-on-Task
(min)',
    'i-Ready Overall: Lessons Passed', 'i-Ready Overall: Lessons
Completed',
    'i-Ready Overall: % Lessons Passed',
    'i-Ready Pro Overall: Lessons Completed', 'i-Ready Pro Overall:
Skills Successful',
    'i-Ready Pro Overall: Skills Completed', 'i-Ready Pro Overall: %
Skills Successful',
    'i-Ready Algebra and Algebraic Thinking: Lessons Passed',
    'i-Ready Algebra and Algebraic Thinking: Lessons Completed',
    'i-Ready Algebra and Algebraic Thinking: % Lessons Passed',
    'i-Ready Number and Operations: Lessons Passed',
    'i-Ready Number and Operations: Lessons Completed',
    'i-Ready Number and Operations: % Lessons Passed',
    'i-Ready Measurement and Data: Lessons Passed',
```

```
    'i-Ready Measurement and Data: Lessons Completed',
    'i-Ready Measurement and Data: % Lessons Passed',
    'i-Ready Geometry: Lessons Passed',
    'i-Ready Geometry: Lessons Completed',
    'i-Ready Geometry: % Lessons Passed'
]

def fetch_ai_insights(data_point):
    """Simulate fetching AI insights for a given data point."""
    return {
        "explanation": f"This metric represents {data_point} and its
impact on overall student performance.",
        "suggestions": [
            f"Consider analyzing {data_point} in relation to i-Ready
scores.",
            f"Explore historical trends for {data_point} to identify
patterns."
        ]
    }

def filter_non_attempted_students(data, domain_columns):
    """Filter out students who have not attempted any lessons in a
domain."""
    return data[(data[domain_columns[0]] > 0) |
(data[domain_columns[1]] > 0) | (data[domain_columns[2]].notnull())]

if uploaded_file:
    # Read the CSV file
    data = pd.read_csv(uploaded_file)

    # Check for required columns
    missing_columns = [col for col in required_columns if col not in
data.columns]
    if missing_columns:
        st.error(f"The following required columns are missing from the
uploaded file: {', '.join(missing_columns)}")
        st.stop()

    # Display the first few rows of the dataset
    st.subheader("Preview of Uploaded Data")
    st.dataframe(data.head())

    # Sidebar filters
    st.sidebar.header("Filters")
    student_grade_filter = st.sidebar.multiselect(
        "Select Student Grades", options=data['Student
Grade'].unique(), default=data['Student Grade'].unique()
    )
    class_teacher_filter = st.sidebar.multiselect(
        "Select Class Teachers", options=data['Class
```

```
Teacher(s)'].unique(), default=data['Class Teacher(s)'].unique()
    )
    subject_filter = st.sidebar.multiselect(
        "Select Subjects", options=data['Subject'].unique(),
default=data['Subject'].unique()
    )
    school_filter = st.sidebar.multiselect(
        "Select Schools", options=data['School'].unique(),
default=data['School'].unique()
    )

    filtered_data = data[
        (data['Student Grade'].isin(student_grade_filter)) &
        (data['Class Teacher(s)'].isin(class_teacher_filter)) &
        (data['Subject'].isin(subject_filter)) &
        (data['School'].isin(school_filter))
    ]

    # Show filtered data
    st.subheader("Filtered Data")
    st.dataframe(filtered_data)

    # Experimental Component: Overview
    st.subheader("Experimental Components")  # Line 109
    if st.checkbox("Show Data Overview"):    # Line 110
     overview_component(filtered_data)    # Line 111

    # Key Metrics for i-Ready
    st.subheader("Key Metrics Analysis: i-Ready")
    iready_columns = [
        'Total Lesson Time-on-Task (min)',
        'i-Ready Overall: Lessons Passed',
        'i-Ready Overall: Lessons Completed',
        'i-Ready Overall: % Lessons Passed'
    ]
    iready_data = filtered_data[iready_columns]

    st.sidebar.header("i-Ready Visualization Options")
    iready_vis_type = st.sidebar.selectbox("Visualization Type for i-
Ready", ["Scatter Plot", "Bar Chart", "Box Plot", "Correlation
Heatmap"], key="iready_vis")

    if iready_vis_type == "Scatter Plot":
        col_x = st.sidebar.selectbox("X-axis", options=iready_columns,
key="iready_x")
        col_y = st.sidebar.selectbox("Y-axis", options=iready_columns,
key="iready_y")
        fig, ax = plt.subplots()
        sns.scatterplot(data=iready_data, x=col_x, y=col_y, ax=ax)
        ax.set_title(f"Scatter Plot: {col_x} vs {col_y}")
```

```
        st.pyplot(fig)

        if st.button("Show AI Insights", key=f"{col_x}-{col_y}-
insights"):
            insights = fetch_ai_insights(col_x)
            st.info(f"Explanation: {insights['explanation']}")
            st.write("Suggestions:")
            for suggestion in insights['suggestions']:
                st.write(f"- {suggestion}")

    elif iready_vis_type == "Bar Chart":
        avg_values = iready_data.mean()
        st.bar_chart(avg_values)

        if st.button("Show AI Insights", key="iready-bar-chart-
insights"):
            insights = fetch_ai_insights("i-Ready Metrics (Bar
Chart)")
            st.info(f"Explanation: {insights['explanation']}")
            st.write("Suggestions:")
            for suggestion in insights['suggestions']:
                st.write(f"- {suggestion}")

    elif iready_vis_type == "Box Plot":
        fig, ax = plt.subplots()
        sns.boxplot(data=iready_data, ax=ax)
        ax.set_title("Distribution of i-Ready Metrics")
        st.pyplot(fig)

        if st.button("Show AI Insights", key="iready-box-plot-
insights"):
            insights = fetch_ai_insights("i-Ready Metrics (Box Plot)")
            st.info(f"Explanation: {insights['explanation']}")
            st.write("Suggestions:")
            for suggestion in insights['suggestions']:
                st.write(f"- {suggestion}")

    elif iready_vis_type == "Correlation Heatmap":
        fig, ax = plt.subplots()
        sns.heatmap(iready_data.corr(), annot=True, cmap="coolwarm",
ax=ax)
        ax.set_title("Correlation Heatmap of i-Ready Metrics")
        st.pyplot(fig)

        if st.button("Show AI Insights", key="iready-heatmap-
insights"):
            insights = fetch_ai_insights("i-Ready Metrics (Correlation
Heatmap)")
            st.info(f"Explanation: {insights['explanation']}")
            st.write("Suggestions:")
```

```
            for suggestion in insights['suggestions']:
                st.write(f"- {suggestion}")

    # Domain Analysis
    st.subheader("Domain Analysis")
    domains = {
        "Algebra and Algebraic Thinking": [
            'i-Ready Algebra and Algebraic Thinking: Lessons Passed',
            'i-Ready Algebra and Algebraic Thinking: Lessons
Completed',
            'i-Ready Algebra and Algebraic Thinking: % Lessons Passed'
        ],
        "Number and Operations": [
            'i-Ready Number and Operations: Lessons Passed',
            'i-Ready Number and Operations: Lessons Completed',
            'i-Ready Number and Operations: % Lessons Passed'
        ],
        "Measurement and Data": [
            'i-Ready Measurement and Data: Lessons Passed',
            'i-Ready Measurement and Data: Lessons Completed',
            'i-Ready Measurement and Data: % Lessons Passed'
        ],
        "Geometry": [
            'i-Ready Geometry: Lessons Passed',
            'i-Ready Geometry: Lessons Completed',
            'i-Ready Geometry: % Lessons Passed'
        ]
    }

    selected_domain = st.selectbox("Select Domain for Analysis",
options=list(domains.keys()))
    domain_columns = domains[selected_domain]

    # Filter out students who haven't attempted any lessons in the
selected domain
    filtered_comparison_data =
filter_non_attempted_students(filtered_data, domain_columns)

    st.sidebar.header(f"{selected_domain} Analysis Options")
    pass_threshold = st.sidebar.slider("Select % Passed Threshold",
min_value=0, max_value=100, value=50)
    failing_students =
filtered_comparison_data[filtered_comparison_data[domain_columns[2]] <
pass_threshold]

    st.write(f"### Students with {selected_domain} % Passed Below
{pass_threshold}%")
    st.dataframe(failing_students[["Last Name", "First Name", "Student
ID"] + domain_columns])
```

```
    # Class vs Other Classes Comparison - Grouped Bar Chart
    st.write("### Class Progress vs Other Selected Classes")
    class_avg = filtered_comparison_data.groupby("Class Teacher(s)")
[domain_columns].mean().reset_index()
    melted_data = class_avg.melt(id_vars="Class Teacher(s)",
var_name="Metric", value_name="Value")
    fig = px.bar(
        melted_data,
        x="Class Teacher(s)",
        y="Value",
        color="Metric",
        title=f"{selected_domain} - Class Progress vs Selected
Classes",
        barmode="group"
    )
    st.plotly_chart(fig)

    if st.button("Show AI Insights", key="domain-bar-chart-insights"):
        insights = fetch_ai_insights(f"{selected_domain} Domain Bar
Chart")
        st.info(f"Explanation: {insights['explanation']}")
        st.write("Suggestions:")
        for suggestion in insights['suggestions']:
            st.write(f"- {suggestion}")

    # Student Progress Distribution
    st.write("### Student Progress Distribution")
    fig, ax = plt.subplots()
    sns.histplot(filtered_comparison_data[domain_columns[2]], bins=10,
kde=True, ax=ax)
    ax.set_title(f"Distribution of % Lessons Passed in
{selected_domain}")
    st.pyplot(fig)

    if st.button("Show AI Insights", key="domain-distribution-
insights"):
        insights = fetch_ai_insights(f"{selected_domain} Progress
Distribution")
        st.info(f"Explanation: {insights['explanation']}")
        st.write("Suggestions:")
        for suggestion in insights['suggestions']:
            st.write(f"- {suggestion}")

    # Correlation within Domain
    st.write("### Correlation Analysis for Domain Metrics")
    fig, ax = plt.subplots()
    sns.heatmap(filtered_comparison_data[domain_columns].corr(),
annot=True, cmap="coolwarm", ax=ax)
    st.pyplot(fig)
```

```
    if st.button("Show AI Insights", key="domain-correlation-heatmap-
insights"):
        insights = fetch_ai_insights(f"{selected_domain} Correlation
Heatmap")
        st.info(f"Explanation: {insights['explanation']}")
        st.write("Suggestions:")
        for suggestion in insights['suggestions']:
            st.write(f"- {suggestion}")

else:
    st.write("Please upload a CSV file to start analyzing data.")

# ========================================
#              NEW FEATURES
# ========================================

def overview_component(data):
    """
    Display an overview of the dataset, including column stats and
unique counts.
    """
    st.write("## Data Overview")
    st.write(f"Number of rows: {data.shape[0]}")
    st.write(f"Number of columns: {data.shape[1]}")
    st.write("Columns and unique values:")
    unique_counts = {col: data[col].nunique() for col in data.columns}
    st.write(unique_counts)
```