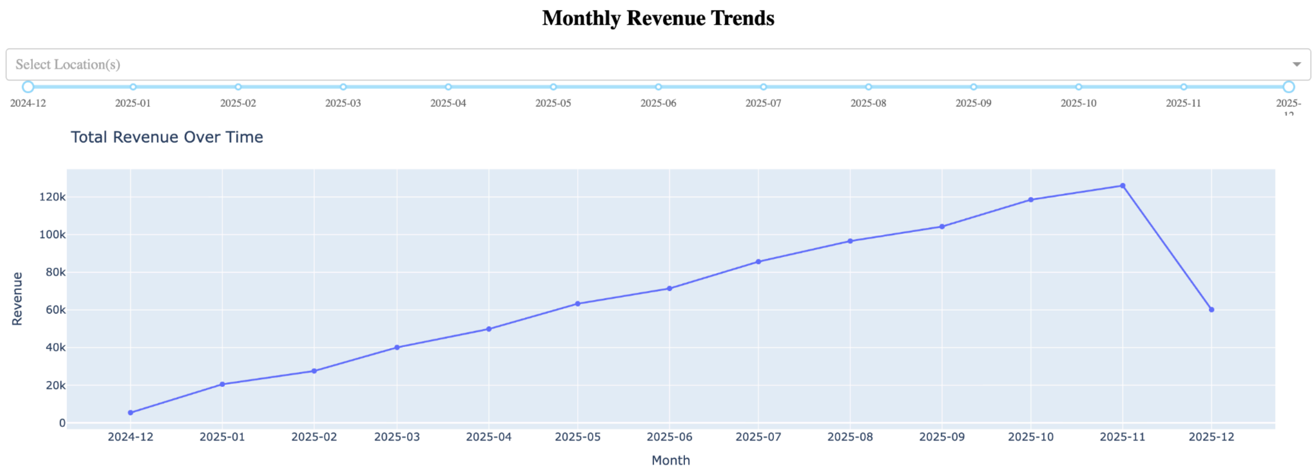
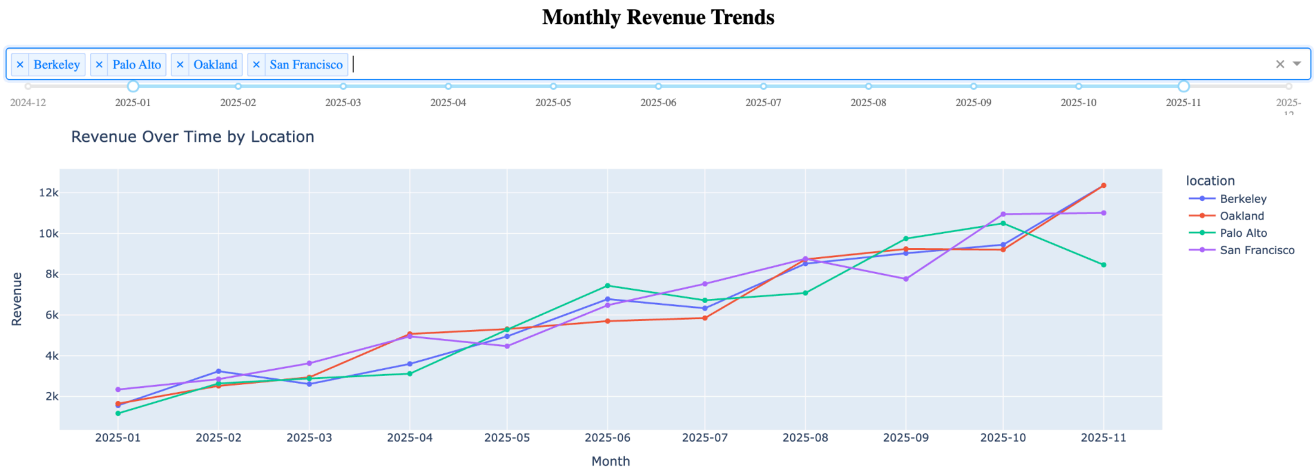
FitFlow Management Dashboard

Plot 1: Monthly Revenue Trends

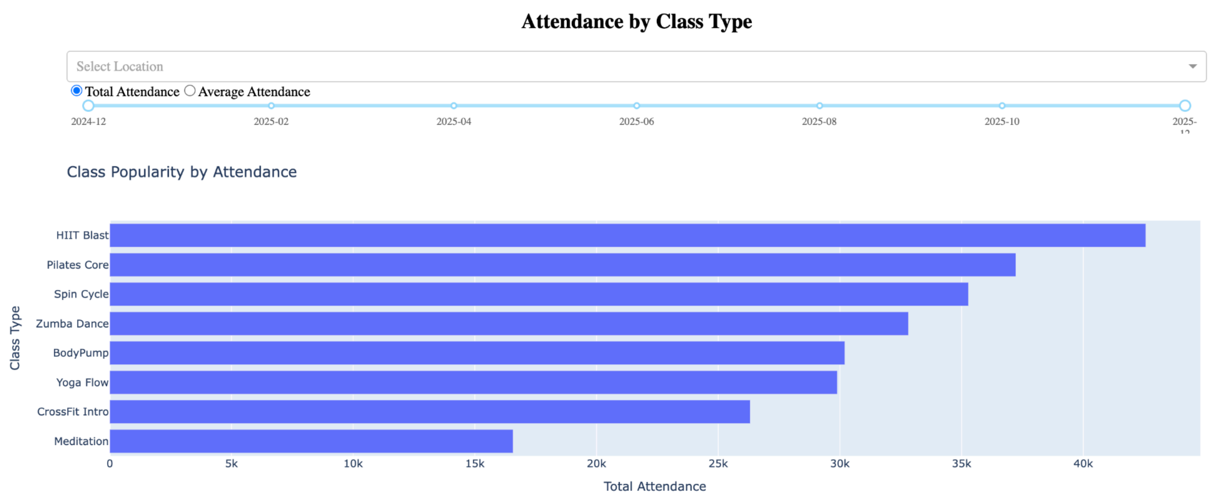


Plot 1.1: Monthly Revenue Trends *filtered by locations (Berkeley, Palo Alto, Oakland, and San Francisco) and range of months*



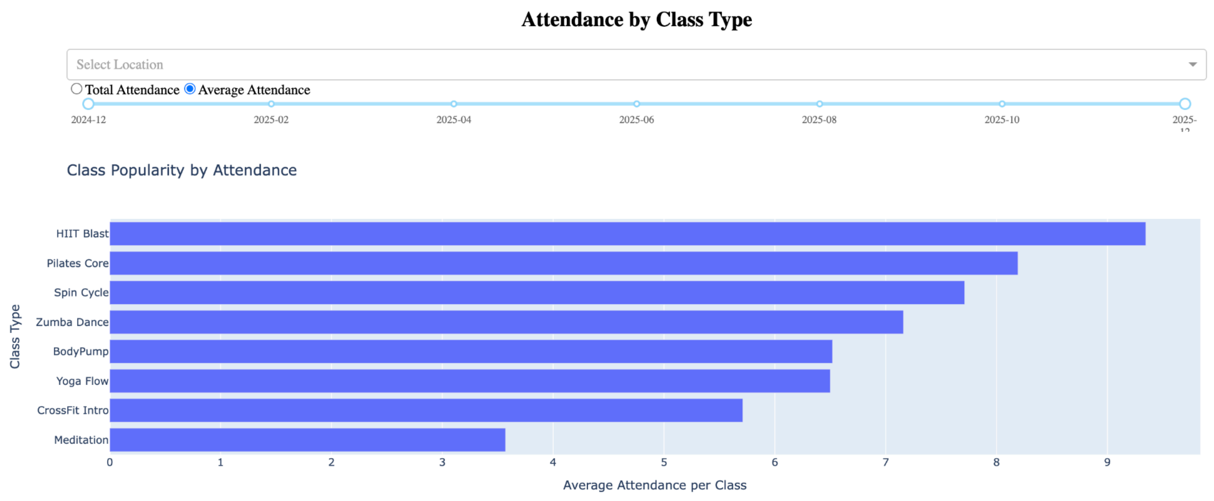
Note: When no location is selected, the chart aggregates revenue across all locations to provide an overall business view.

Plot 2: Total Attendance by Class Type


Select Location

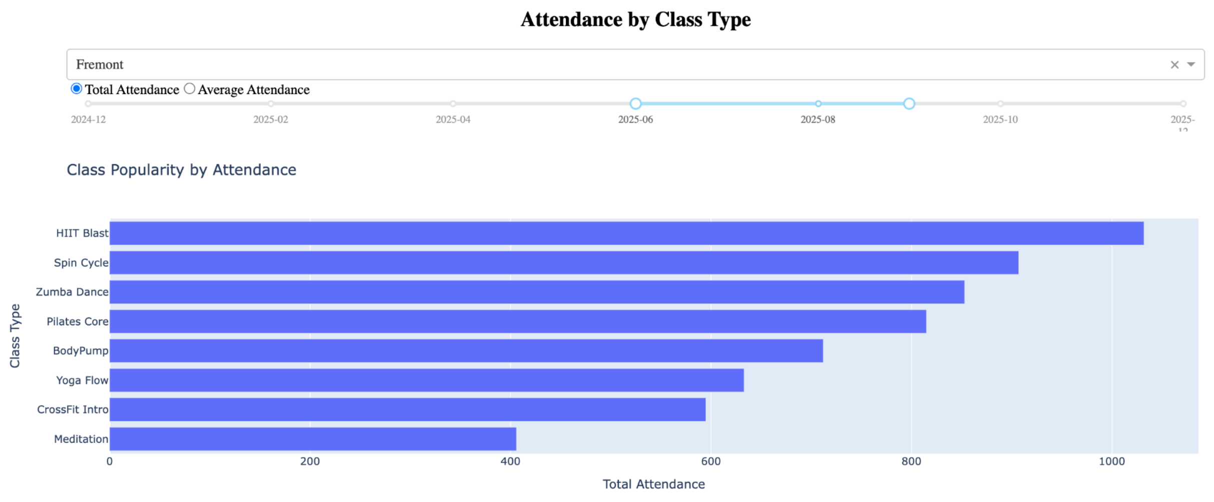
[✓ Total Attendance] Average Attendance

Plot 2.1: Average Attendance by Class Type


Select Location

Total Attendance [✓ Average Attendance]

Plot 2.2: Total Attendance by Class Type *filtered by location (Fremont) and months (Q3, 2025)*

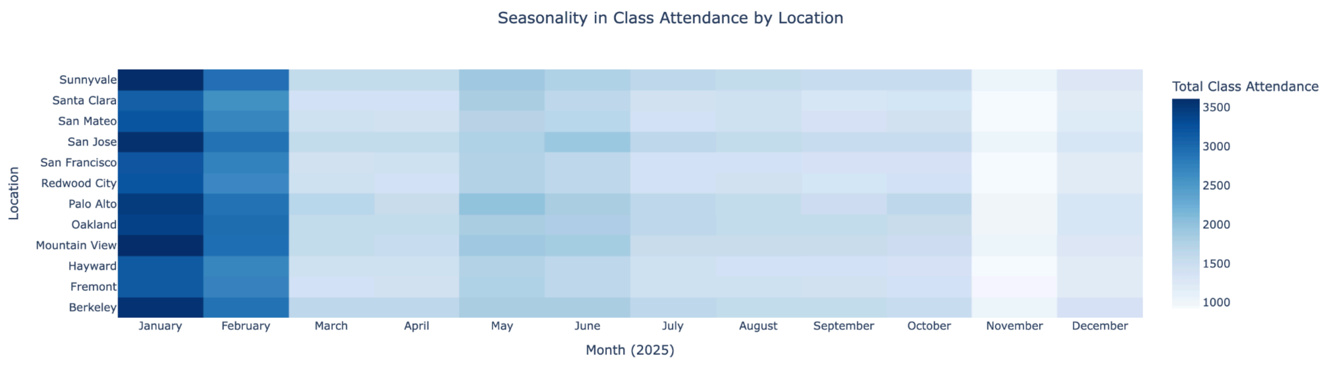

Fremont

[✓ Total Attendance] Average Attendance

Plot 3: Seasonality Analysis of Class Attendance

Seasonality Analysis: Attendance Heatmap

Select a location (optional)

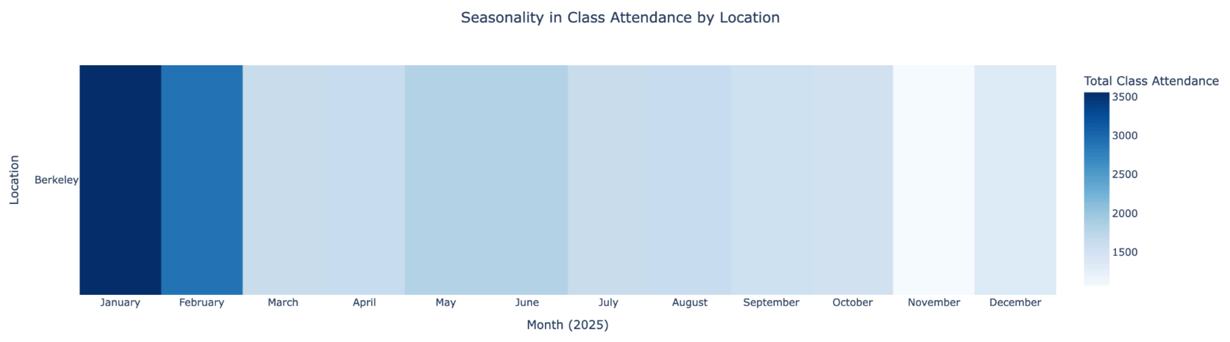


Darker colors indicate higher attendance. January spikes highlight new-year participation behavior.

Plot 3.1: Seasonality Analysis of Class Attendance *filtered by location (Berkeley)*

Seasonality Analysis: Attendance Heatmap

Berkeley



Darker colors indicate higher attendance. January spikes highlight new-year participation behavior.

Appendix

```
#%%

# =====================================================
# IMPORTS
# =====================================================
import pandas as pd
import urllib
from sqlalchemy import create_engine

import dash
from dash import dcc, html
from dash.dependencies import Input, Output
import plotly.express as px

# =====================================================
# DB CONNECTION
# =====================================================
server = 'sql-fitflow-team8-2025.database.windows.net'
database = 'fitflow_olap'
username = 'fitflowadmin'
password = 'Kergerisgreat2025!'

params = urllib.parse.quote_plus(
   f"Driver={{ODBC Driver 18 for SQL Server}};"
   f"Server=tcp:{server},1433;"
   f"Database={database};"
   f"Uid={username};Pwd={password};"
   f"Encrypt=yes;TrustServerCertificate=no;Connection Timeout=30;"
)

engine = create_engine(f"mssql+pyodbc:///?odbc_connect={params}")

# =====================================================
# LOAD DATA — VISUALIZATION 1 (REVENUE)
# =====================================================
query_revenue = """
SELECT
   d.year,
   d.month,
   CONCAT(d.year, '-', RIGHT('00' + CAST(d.month AS VARCHAR), 2)) AS year_month,
   l.city AS location,
   SUM(f.total_revenue) AS revenue
FROM Fact_Monthly_Revenue f
JOIN Dim_Date d ON f.date_key = d.date_key
JOIN Dim_Location l ON f.location_key = l.location_key
GROUP BY d.year, d.month, l.city
ORDER BY d.year, d.month;
"""

df_rev = pd.read_sql(query_revenue, engine)
months_rev = sorted(df_rev["year_month"].unique())

# ======================================================
# LOAD DATA — VISUALIZATION 2 (CLASS ATTENDANCE)
# ======================================================
query_attendance = """
SELECT
   d.year,
   d.month,
   CONCAT(d.year, '-', RIGHT('00' + CAST(d.month AS VARCHAR), 2)) AS year_month,
   l.city AS location,
   c.name AS class_type,
   SUM(f.attendance_count) AS total_attendance,
   AVG(f.attendance_count) AS avg_attendance
FROM Fact_Class_Attendance f
JOIN Dim_Date d ON f.date_key = d.date_key
JOIN Dim_Location l ON f.location_key = l.location_key
JOIN Dim_Class_Type c ON f.class_type_key = c.class_type_key
GROUP BY d.year, d.month, l.city, c.name
ORDER BY d.year, d.month;
"""

df_att = pd.read_sql(query_attendance, engine)
months_att = sorted(df_att["year_month"].unique())

# ======================================================
# LOAD DATA — VISUALIZATION 3 (HEATMAP)
# ======================================================
query_heatmap = """
SELECT
   DATENAME(month, DATEFROMPARTS(d.year, d.month, 1)) AS month_name,
   l.city AS location,
   SUM(f.attendance_count) AS total_attendance
FROM Fact_Class_Attendance f
JOIN Dim_Date d ON f.date_key = d.date_key
JOIN Dim_Location l ON f.location_key = l.location_key
GROUP BY d.year, d.month, l.city
"""

df_heat = pd.read_sql(query_heatmap, engine)

month_order = [
   "January","February","March","April","May","June",
   "July","August","September","October","November","December"
]

df_heat["month_name"] = pd.Categorical(
   df_heat["month_name"],
   categories=month_order,
   ordered=True
)

# =====================================================
# DASH APP (ONE APP ONLY)
# =====================================================
app = dash.Dash(__name__)

app.layout = html.Div([

   html.H1("FitFlow Management Dashboard", style={"textAlign": "center"}),

   # =================================================
   # VISUALIZATION 1 — REVENUE
   # =================================================
   html.H2("Monthly Revenue Trends"),

   dcc.Dropdown(
       id="rev-location-filter",
       options=[{"label": i, "value": i} for i in
sorted(df_rev["location"].unique())],
       multi=True,
       placeholder="Select Location(s)"
   ),
   dcc.RangeSlider(
       id="rev-time-filter",
       min=0,
       max=len(months_rev) - 1,
       value=[0, len(months_rev) - 1],
       marks={i: months_rev[i] for i in range(len(months_rev))},
       step=1
   ),

   dcc.Graph(id="revenue-line"),

   html.Hr(),

   # ==================================================
   # VISUALIZATION 2 — CLASS ATTENDANCE
   # ==================================================
   html.H2("Attendance by Class Type"),

   dcc.Dropdown(
       id="att-location-filter",
       options=[{"label": i, "value": i} for i in
sorted(df_att["location"].unique())],
       placeholder="Select Location",
       clearable=True
   ),

   dcc.RadioItems(
       id="metric-toggle",
       options=[
           {"label": "Total Attendance", "value": "total"},
           {"label": "Average Attendance", "value": "average"}
       ],
       value="total",
       inline=True
   ),

   dcc.RangeSlider(
       id="att-time-filter",
       min=0,
       max=len(months_att) - 1,
       value=[0, len(months_att) - 1],
       marks={i: months_att[i] for i in range(0, len(months_att), max(1,
len(months_att)//6))},
       step=1
   ),

   dcc.Graph(id="attendance-bar"),

   html.Hr(),

   # ==================================================
   # VISUALIZATION 3 — HEATMAP
   # ==================================================
   html.H2("Seasonality in Class Attendance"),

   dcc.Dropdown(
       id="heat-location-filter",
       options=[{"label": loc, "value": loc} for loc in
sorted(df_heat["location"].unique())],
       placeholder="Select Location (optional)",
       clearable=True,
       style={"width": "40%"}
   ),

   dcc.Graph(id="attendance-heatmap")
])

# =====================================================
# CALLBACKS
# =====================================================

@app.callback(
   Output("revenue-line", "figure"),
   Input("rev-location-filter", "value"),
   Input("rev-time-filter", "value")
)
def update_revenue(locations, time_range):
   dff = df_rev.copy()
   dff = dff[dff["year_month"].between(
       months_rev[time_range[0]], months_rev[time_range[1]]
   )]

   if locations:
       fig = px.line(dff[dff["location"].isin(locations)],
                     x="year_month", y="revenue",
                     color="location", markers=True)
   else:
       dff = dff.groupby("year_month", as_index=False)["revenue"].sum()
       fig = px.line(dff, x="year_month", y="revenue", markers=True)

   fig.update_xaxes(categoryorder="array", categoryarray=months_rev)
   fig.update_layout(xaxis_title="Month", yaxis_title="Revenue")
   return fig


@app.callback(
   Output("attendance-bar", "figure"),
   Input("att-location-filter", "value"),
   Input("metric-toggle", "value"),
   Input("att-time-filter", "value")
)
def update_attendance(location, metric, time_range):
   dff = df_att.copy()
   dff = dff[dff["year_month"].between(
       months_att[time_range[0]], months_att[time_range[1]]
   )]

   if location:
       dff = dff[dff["location"] == location]

   if metric == "total":
       dff = dff.groupby("class_type", as_index=False)["total_attendance"].sum()
       y = "total_attendance"
   else:
       dff = dff.groupby("class_type", as_index=False)["avg_attendance"].mean()
       y = "avg_attendance"

   fig = px.bar(dff, x=y, y="class_type", orientation="h")
   return fig


@app.callback(
   Output("attendance-heatmap", "figure"),
   Input("heat-location-filter", "value")
)
```

```python
def update_heatmap(location):
    dff = df_heat.copy()
    if location:
        dff = dff[dff["location"] == location]

    fig = px.density_heatmap(
        dff,
        x="month_name",
        y="location",
        z="total_attendance",
        color_continuous_scale="Blues"
    )

    fig.update_coloraxes(colorbar_title="Total Class Attendance")
    return fig


# ======================================================
# RUN
# ======================================================
if __name__ == "__main__":
    app.run(debug=True)

# dash: http://127.0.0.1:8050/
```