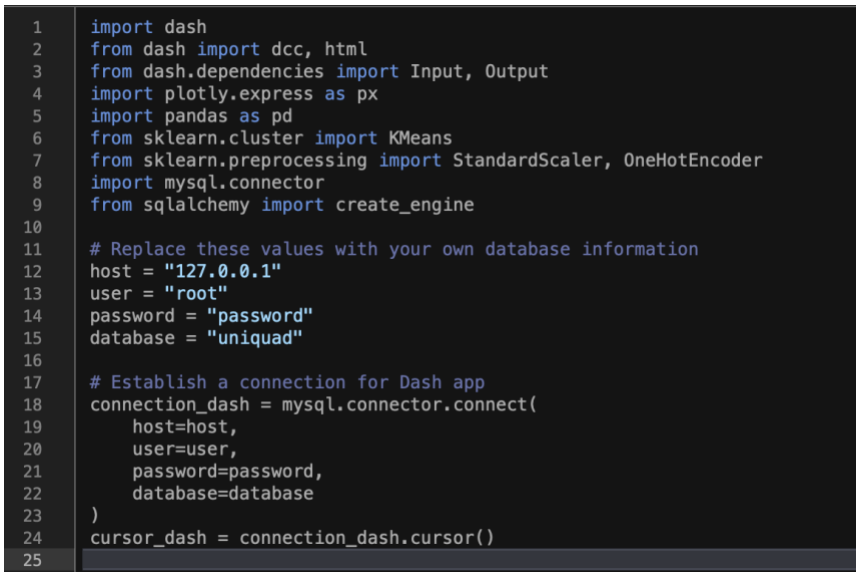
Query execution using Python :

Connected MySQL database using **mysql.connector** and executed the following queries.



```
import dash
from dash import dcc, html
from dash.dependencies import Input, Output
import plotly.express as px
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler, OneHotEncoder
import mysql.connector
from sqlalchemy import create_engine

# Replace these values with your own database information
host = "127.0.0.1"
user = "root"
password = "password"
database = "uniquad"

# Establish a connection for Dash app
connection_dash = mysql.connector.connect(
    host=host,
    user=user,
    password=password,
    database=database
)
cursor_dash = connection_dash.cursor()
```

Query 1 :

```
query1 = """
   SELECT
      o.name AS Organization_Name,
      COUNT(os.std_id) AS Student_Count
   FROM
      organization o
   JOIN
      student_in_organization os ON o.Id = os.Org_id
   GROUP BY
      o.name;
"""
print("Executing Query 1:")
print(query1)
cursor_dash.execute(query1)
results1 = cursor_dash.fetchall()
print("Query 1 Results:")
for row in results1:
   organization_name, student_count = row
   print(f"Organization Name: {organization_name}, Student Count: {student_count}")
print("\n")
```

Output of query 1:

```
Executing Query 1:

    SELECT
        o.name AS Organization_Name,
        COUNT(os.std_id) AS Student_Count
    FROM
        organization o
    JOIN
        student_in_organization os ON o.Id = os.Org_id
    GROUP BY
        o.name;

Query 1 Results:
Organization Name: Future Innovators Society (FIS), Student Count: 3
Organization Name: Global Perspectives Club (GPC), Student Count: 3
Organization Name: Tech Wizards Alliance (TWA), Student Count: 3
Organization Name: Environmental Stewards Collective (ESC), Student Count: 4
Organization Name: Literary Explorers Guild (LEG), Student Count: 3
Organization Name: Creative Arts Fusion (CAF), Student Count: 3
Organization Name: Healthy Living Coalition (HLC), Student Count: 3
Organization Name: Entrepreneurship Ambassadors (EA), Student Count: 3
Organization Name: Cultural Exchange Network (CEN), Student Count: 3
Organization Name: STEM Mavericks Society (SMS), Student Count: 5
Organization Name: Mindfulness Matters Circle (MMC), Student Count: 3
```

Similarly query 2 and 3 were executed.

Output of query 2:

```
Executing Query 2:

    SELECT
        alum_name,
        years_of_experience,
        CASE
            WHEN years_of_experience < 5 THEN 'Early Professional'
            WHEN years_of_experience BETWEEN 5 AND 10 THEN 'Professional'
            WHEN years_of_experience > 10 THEN 'Senior Professional'
        END AS Experience_Category
    FROM
        alum
    ORDER BY
        alum_name;

Query 2 Results:
Alum Name: Ajay Sharma, Experience: 6 years, Category: Professional
Alum Name: Alex Miller, Experience: 8 years, Category: Professional
Alum Name: Andrew White, Experience: 8 years, Category: Professional
Alum Name: Bonnie Chen, Experience: 8 years, Category: Professional
Alum Name: Brian Harris, Experience: 6 years, Category: Professional
Alum Name: David Johnson, Experience: 7 years, Category: Professional
Alum Name: Dylan Morris, Experience: 6 years, Category: Professional
Alum Name: Emily White, Experience: 7 years, Category: Professional
Alum Name: Emma Roberts, Experience: 5 years, Category: Professional
Alum Name: Eric Sommers, Experience: 7 years, Category: Professional
Alum Name: Harry Smith, Experience: 7 years, Category: Professional
Alum Name: Hermonie Miller, Experience: 6 years, Category: Professional
Alum Name: Isabella Miller, Experience: 5 years, Category: Professional
Alum Name: Jane Smith, Experience: 4 years, Category: Early Professional
Alum Name: Jhon Wick, Experience: 9 years, Category: Professional
Alum Name: Jhonny Bravo, Experience: 8 years, Category: Professional
Alum Name: John Doe, Experience: 5 years, Category: Professional
Alum Name: Jordan Hill, Experience: 6 years, Category: Professional
Alum Name: Kelly Shankar, Experience: 4 years, Category: Early Professional
Alum Name: Kerr Smith, Experience: 7 years, Category: Professional
Alum Name: Kevin Potter, Experience: 5 years, Category: Professional
Alum Name: Lisa Wang, Experience: 6 years, Category: Professional
Alum Name: Liu Wang, Experience: 4 years, Category: Early Professional
Alum Name: Logan Chen, Experience: 9 years, Category: Professional
Alum Name: Logan Lerman, Experience: 6 years, Category: Professional
Alum Name: Luffy Dragon, Experience: 6 years, Category: Professional
Alum Name: Melissa Nguyen, Experience: 4 years, Category: Early Professional
Alum Name: Michael Brown, Experience: 9 years, Category: Professional
Alum Name: Natalie Jones, Experience: 7 years, Category: Professional
Alum Name: Nathan Wang, Experience: 7 years, Category: Professional
Alum Name: Nethra Murugan, Experience: 6 years, Category: Professional
Alum Name: Nobita Sizuka, Experience: 7 years, Category: Professional
Alum Name: Olivia Johnson, Experience: 8 years, Category: Professional
Alum Name: Olivia Taylor, Experience: 8 years, Category: Professional
Alum Name: Sam Jones, Experience: 6 years, Category: Professional
Alum Name: Sarah Kim, Experience: 5 years, Category: Professional
Alum Name: Shelly Jones, Experience: 8 years, Category: Professional
Alum Name: Shin Nguyen, Experience: 9 years, Category: Professional
Alum Name: Shincan Johnson, Experience: 6 years, Category: Professional
Alum Name: Victoria Smith, Experience: 6 years, Category: Professional
```

Output of query 3 :

```
Executing Query 3:

    SELECT
        s.Std_id,
        s.std_name,
        SUM(ib.price) AS total_items_bought_price
    FROM
        student s
    JOIN items_bought_from_sale ib ON s.Std_id = ib.Std_Id
    GROUP BY
        s.Std_id, s.std_name
    HAVING
        total_items_bought_price <= 100;

Query 3 Results:
Student ID: 14, Student Name: Katherine Hall, Total Items Bought Price: $100.00
Student ID: 38, Student Name: Isabella Miller, Total Items Bought Price: $20.00
Student ID: 25, Student Name: Kevin White, Total Items Bought Price: $50.00
Student ID: 29, Student Name: Alexandria Wang, Total Items Bought Price: $70.00
```

For the graps we used combination of plotly, pandas, sklearn, sqlalchemy and used dash server to create dynamic charts that can be interactive and give more detailed visualization of our analysis.

```
# Establish a connection for SQLAlchemy engine
engine = create_engine(f"mysql+mysqlconnector://{user}:{password}@{host}/{database}")
# Load necessary views into Pandas DataFrames
sales_df = pd.read_sql('SELECT * FROM sale_info_view', engine)
user_df = pd.read_sql('SELECT * FROM user', engine)
student_df = pd.read_sql('SELECT * FROM student', engine)
uni_df = pd.read_sql('SELECT * FROM university', engine)
student_program_department_university_df = pd.read_sql('SELECT * FROM student_program_department_university_view', engine)
# Fetch alum data from the database
alum_df = pd.read_sql('SELECT * FROM alum', engine)

# Check the columns in the DataFrame
print(student_program_department_university_df.columns)

# Assuming you have a table named 'mentor_mentee' in your database
mentor_mentee_df = pd.read_sql('SELECT * FROM mentorship', engine)
```

Pandas read was used to read the default tables

```
# Assuming you have a table named 'mentor_mentee' in your database
mentor_mentee_df = pd.read_sql('SELECT * FROM mentorship', engine)

print("Columns before merging mentor_mentee_df:")
print(mentor_mentee_df.columns)

# Merge mentor_mentee_df with student_df to get university information for mentors
mentor_mentee_df = pd.merge(
    mentor_mentee_df,
    student_df[['Std_id', 'Uni_Id']],
    left_on='Std_id',
    right_on='Std_id',
    how='left'
)

# Print columns after merging
print("Columns after merging mentor_mentee_df:")
print(mentor_mentee_df.columns)

# Merge mentor_mentee_df with university_df to get university names for mentors
mentor_mentee_df = pd.merge(
    mentor_mentee_df,
    uni_df[['Uni_id', 'name']],
    left_on='Uni_Id',
    right_on='Uni_id',
    how='left',
    suffixes=('_mentee', '_mentor')
)

# Print columns after the second merge
```

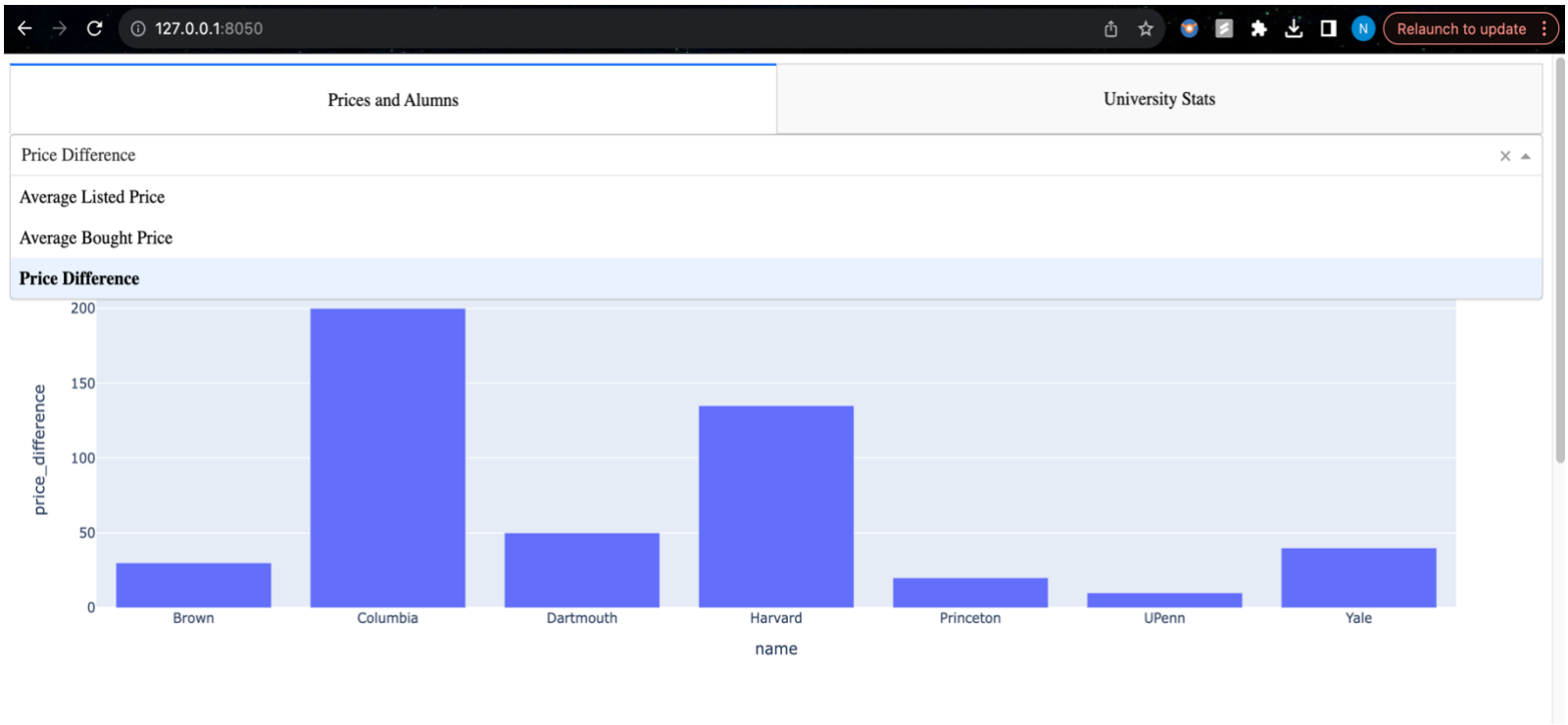
Pandas merge was used whenever we wanted queries to join in python to perform more comprehensive analysis, like above example .Similarly we used a combination of inbuilt views, functions present in our DB and python pandas, plotly and sklearn libraries to perform calculation and other comprehensive analysis to display below visualization, like code in the below example to calculate profit by universities for uniquad app by sales of items through the app and how , we also entire code present in appendix of this assignment and view of the

locally hosted interactive visualization.

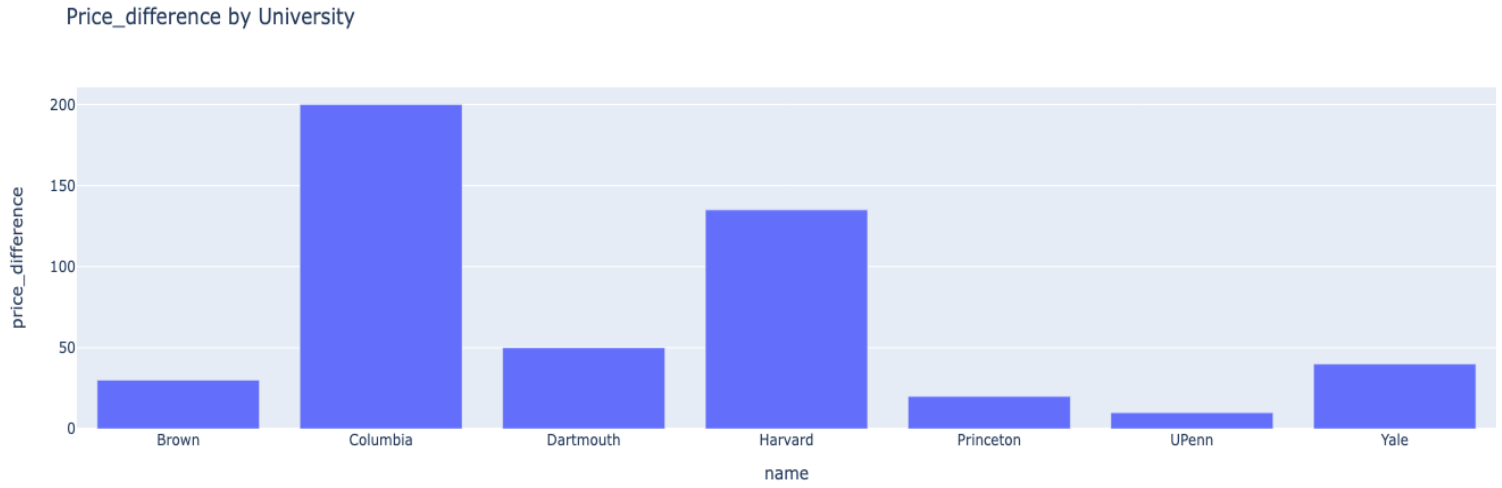
```
    if profit_info.shape[1] == 4:
        # Replace NaN values with a default value
        profit_info = profit_info.fillna(0)

        # Create a dictionary with the extracted values
        university_profit = {
            'university_name': university_name,
            'total_items_sold': int(profit_info.iloc[0, 0]),
            'total_price_obtained': float(profit_info.iloc[0, 1]),
            'total_profit': float(profit_info.iloc[0, 2]),
            'highest_sale': float(profit_info.iloc[0, 3]),
        }

        # Append the dictionary to the list
        universities_profit_data.append(university_profit)
    else:
        print(f"Warning: Unable to extract data for university_name {university_name}")

# Create a DataFrame from the list of dictionaries
universities_by_profit_df = pd.DataFrame(universities_profit_data)

print(student_program_department_university_df['cluster'].unique())

# Initialize Dash app
app = dash.Dash(__name__)

# Define app layout
app.layout = html.Div([
    dcc.Tabs([
        dcc.Tab(label='Prices and Alumns', children=[
            # Dropdown for selecting the metric to display
            dcc.Dropdown(
                id='metric-dropdown',
                options=[
                    {'label': 'Average Listed Price', 'value': 'item_listed_price'},
                    {'label': 'Average Bought Price', 'value': 'item_bought_price'},
                    {'label': 'Price Difference', 'value': 'price_difference'},
                ],
                value='price_difference',
                placeholder='Select Metric'
            ),
```

Graphs output are given below:

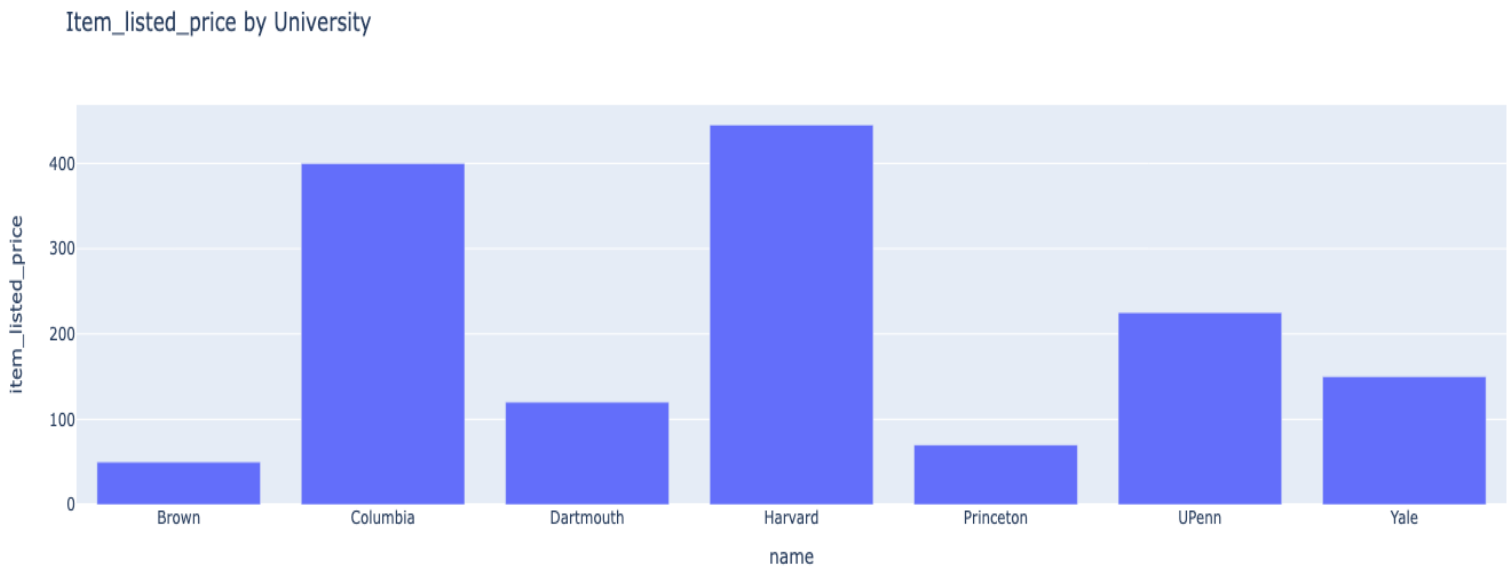
Graph 1: The python code is executed and the following output is obtained. The output of the website has a filter which consists of the following:



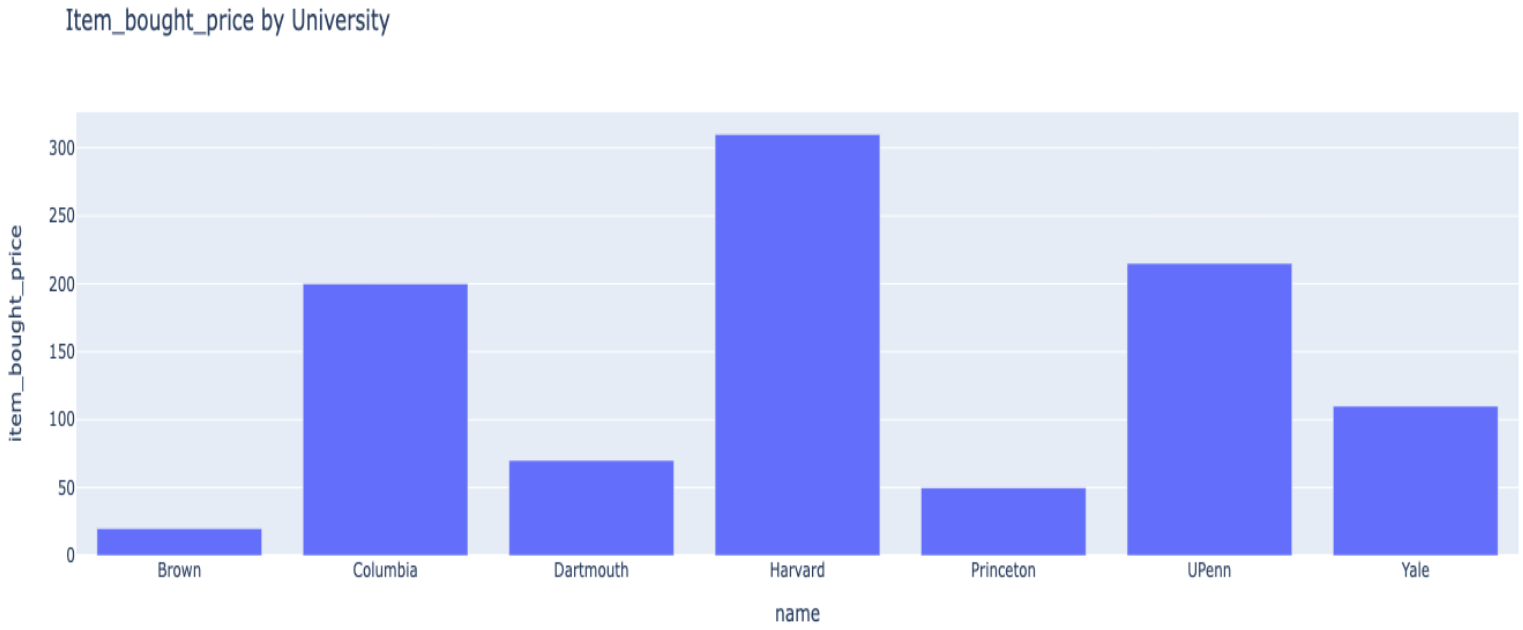
Graph 1.1: Difference in the price of items listed to be sold in each university



Graph 1.2 : Average price of items listed to be sold in each university

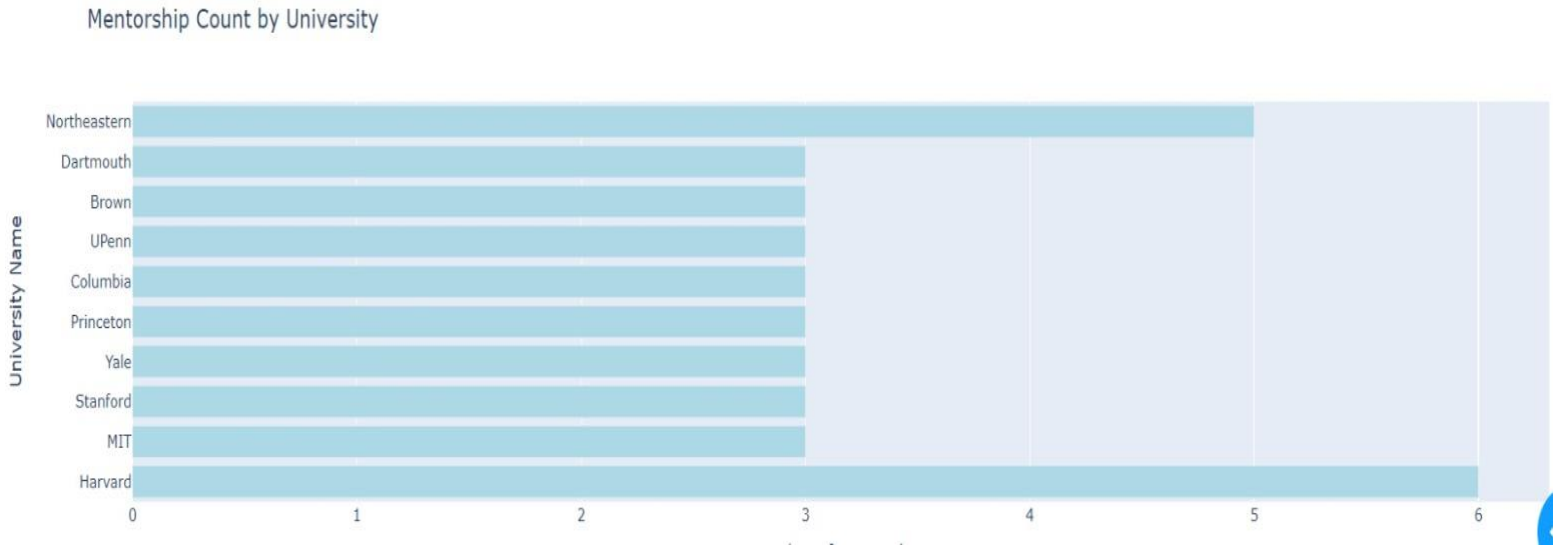


Graph 1.3: Average price of items bought in each university.

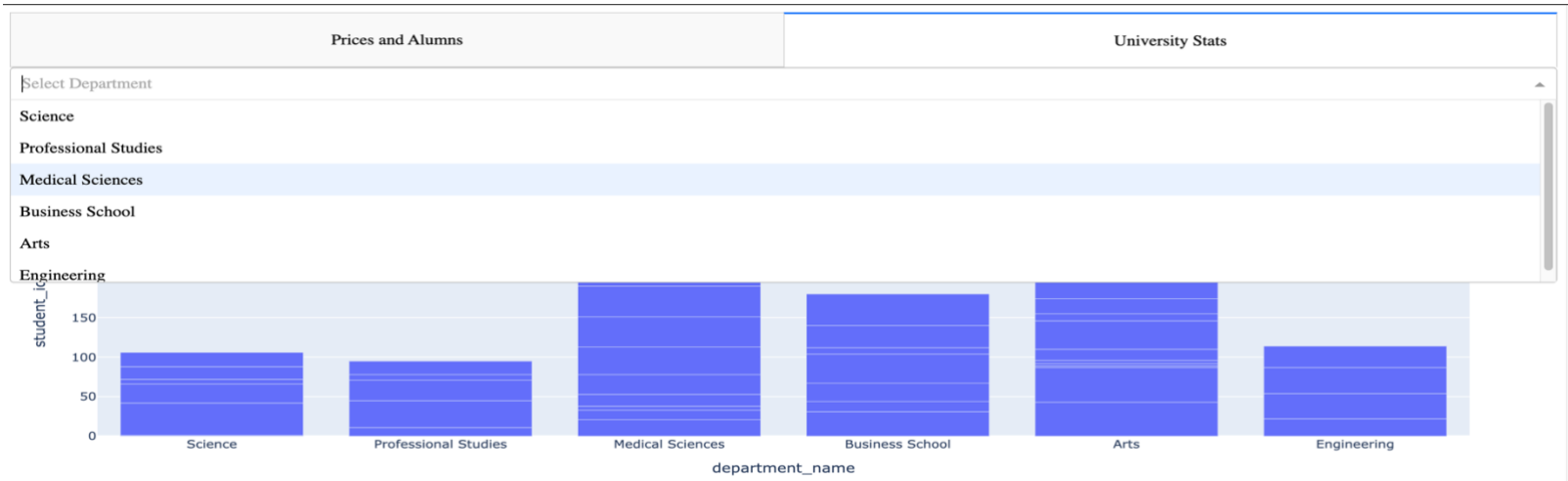


Graph 2: Average alumni experience over time

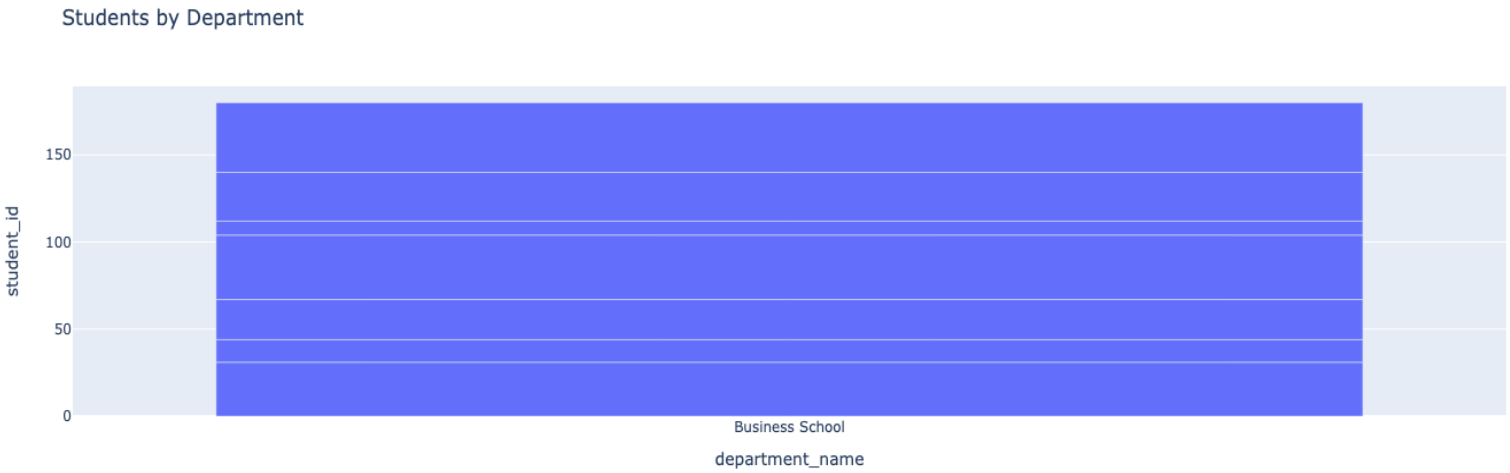
Graph 3: Number of mentors in each university



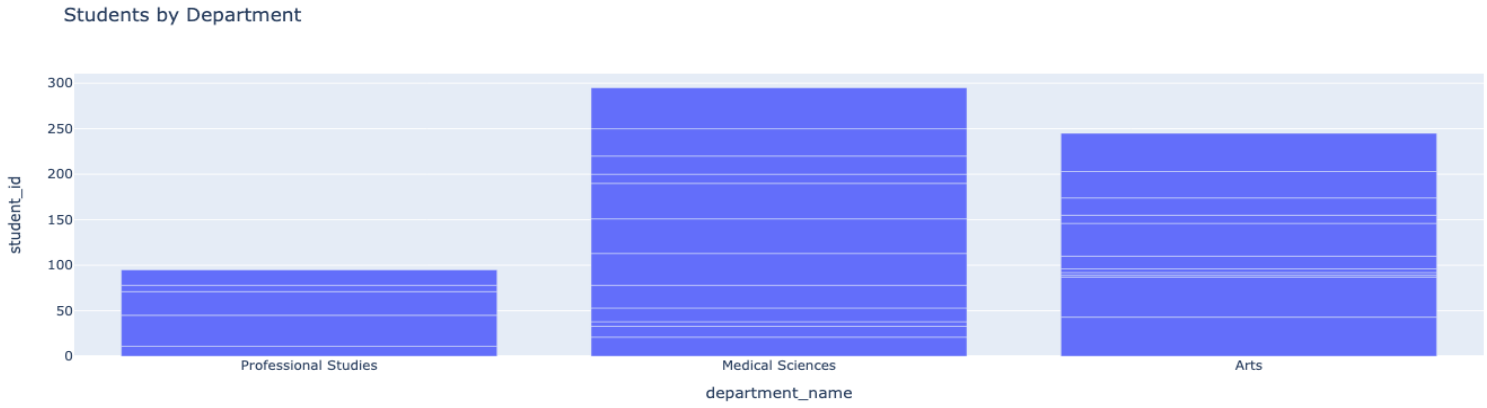
Graph 4 : The bar graph consists of students enrolled in all the departments. There is a filter that retrieves data about no. of students enrolled in a department or compare two.



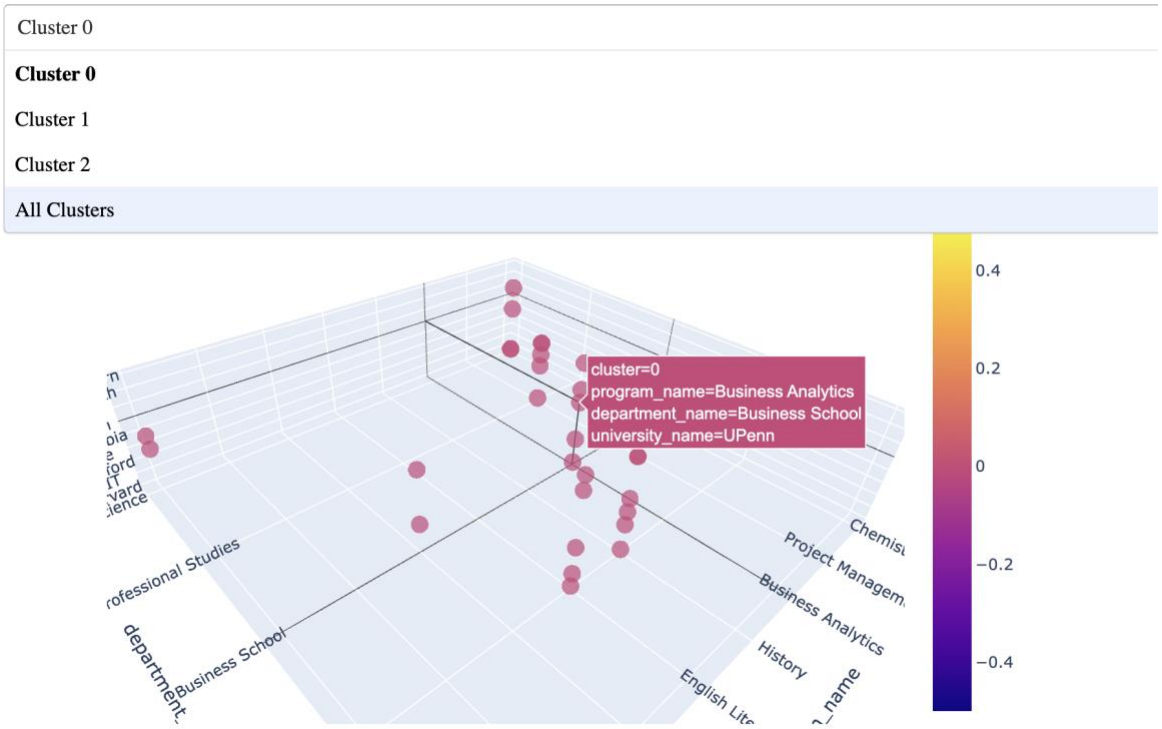
Graph 4.1 : The bar graph displays the number of students enrolled in business school



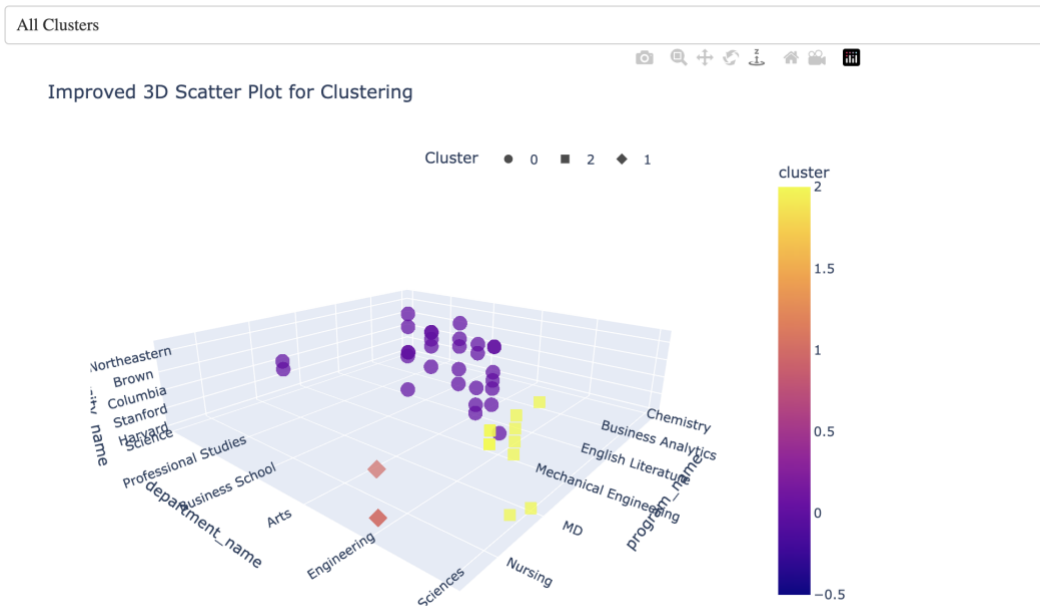
Graph 4.2 : Number of students enrolled in professional studies, medical sciences and arts



Graph 5 : The 3D scatter plot for clustering displays the relation between university, program and department

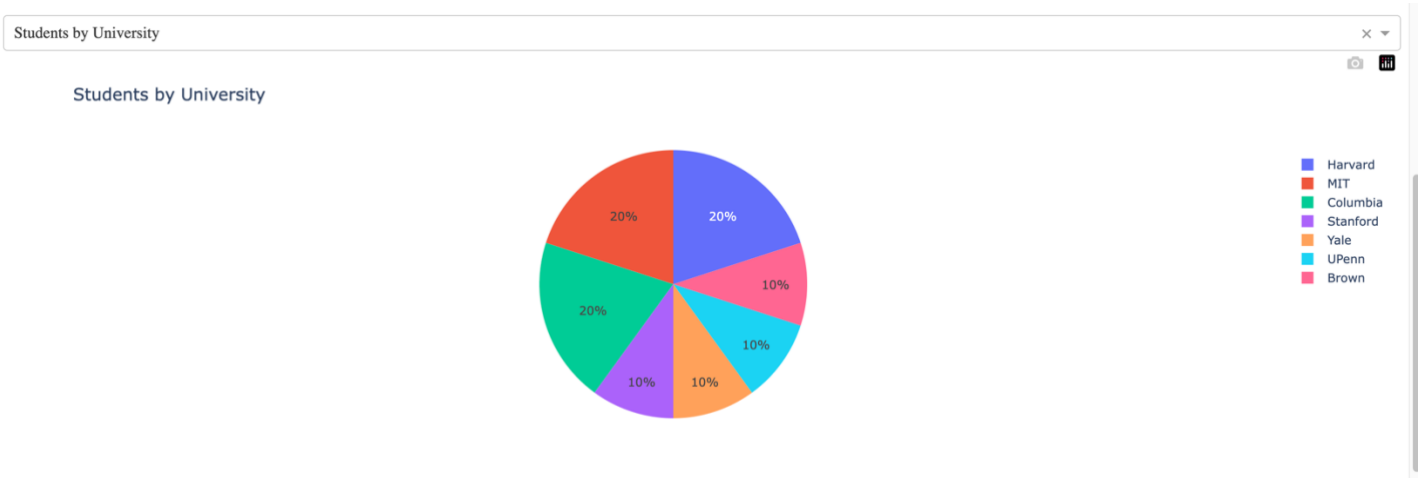


Graph 5.1 : The scatter plot displays all the clusters

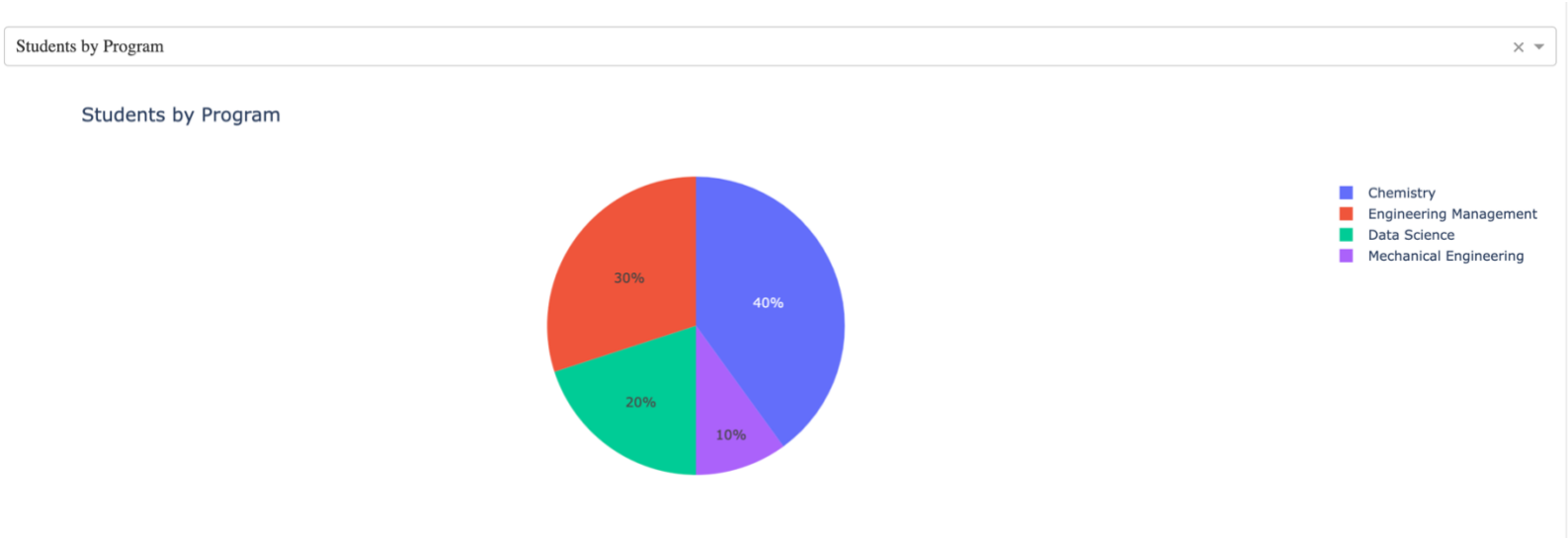


Graph 6 : The pie chart has a filter to display the number of students by university and the number of students by program.

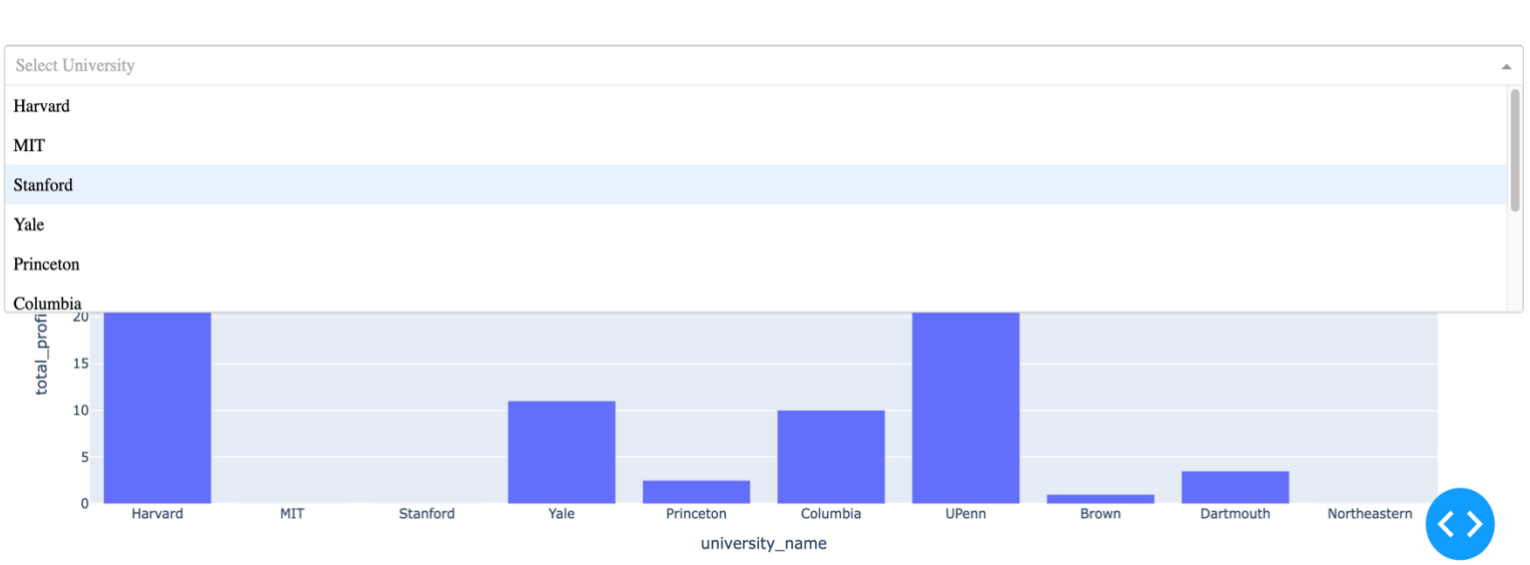
The graph below displays the number of students in the each university.



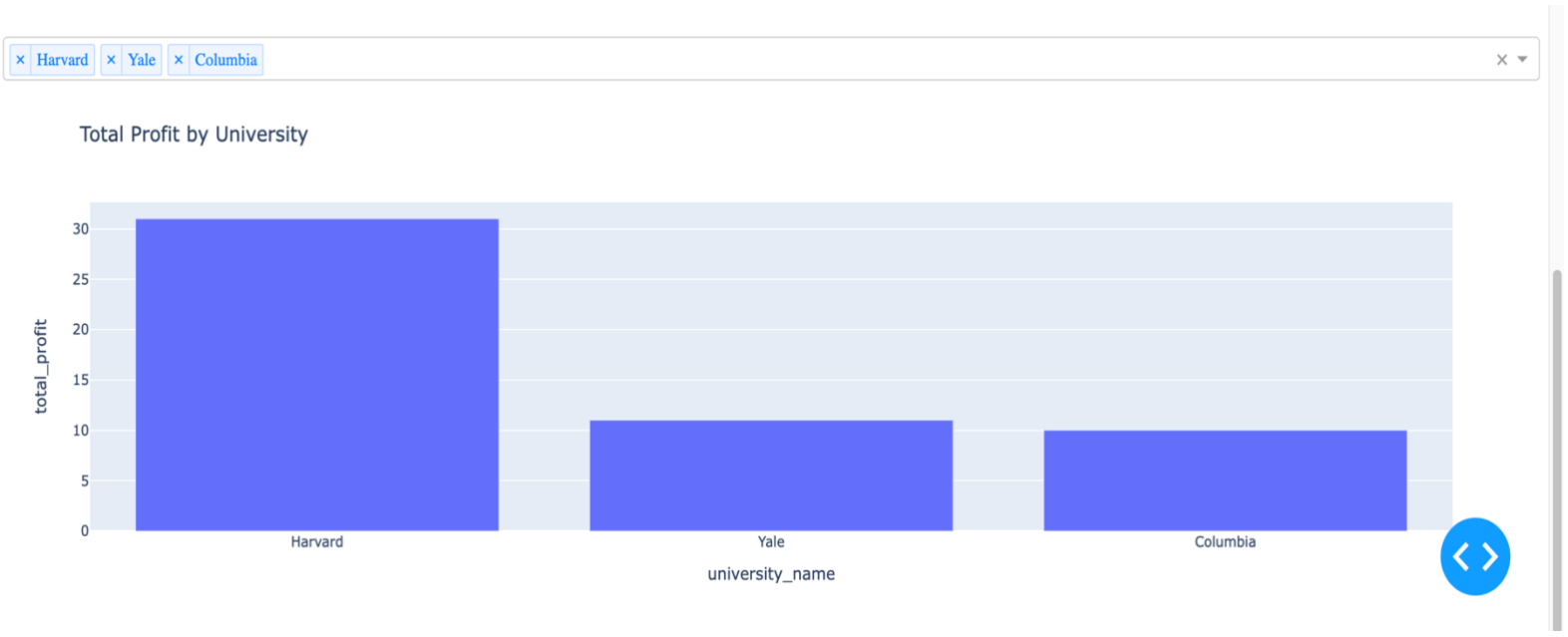
Graph 6.1 : The graph below displays the number of students by program.



Graph 7 : The bar chart displays the total profit uniquad received through the sales of items through the model. The filter has a list of universities to choose and compare from.



Graph 7.1 : The graph compares the total profit between Harvard, Yale and Columbia university.



APPENDIX

CODE:

```
import dash
from dash import dcc, html
from dash.dependencies import Input, Output
import plotly.express as px
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler, OneHotEncoder
import mysql.connector
from sqlalchemy import create_engine

# Replace these values with your own database information
host = "127.0.0.1"
user = "root"
password = "password"
database = "uniquad"

# Establish a connection for Dash app
connection_dash = mysql.connector.connect(
    host=host,
    user=user,
    password=password,
    database=database
)
cursor_dash = connection_dash.cursor()

# Query 1
query1 = """
    SELECT
        o.name AS Organization_Name,
        COUNT(os.std_id) AS Student_Count
    FROM
        organization o
    JOIN
        student_in_organization os ON o.Id = os.Org_id
    GROUP BY
        o.name;
"""
print("Executing Query 1:")
print(query1)
cursor_dash.execute(query1)
results1 = cursor_dash.fetchall()
print("Query 1 Results:")
for row in results1:
    organization_name, student_count = row
    print(f"Organization Name: {organization_name}, Student Count: {student_count}")
print("\n")

# Query 2
query2 = """
```

```
    SELECT
        alum_name,
        years_of_experience,
        CASE
            WHEN years_of_experience < 5 THEN 'Early Professional'
            WHEN years_of_experience BETWEEN 5 AND 10 THEN 'Professional'
            WHEN years_of_experience > 10 THEN 'Senior Professional'
        END AS Experience_Category
    FROM
        alum
    ORDER BY
        alum_name;
"""
print("Executing Query 2:")
print(query2)
cursor_dash.execute(query2)
results2 = cursor_dash.fetchall()
print("Query 2 Results:")
for row in results2:
    alum_name, experience, category = row
    print(f"Alum Name: {alum_name}, Experience: {experience} years, Category: {category}")
print("\n")

# Query 3
query3 = """
    SELECT
        s.Std_id,
        s.std_name,
        SUM(ib.price) AS total_items_bought_price
    FROM
        student s
    JOIN items_bought_from_sale ib ON s.Std_id = ib.Std_Id
    GROUP BY
        s.Std_id, s.std_name
    HAVING
        total_items_bought_price <= 100;
"""
print("Executing Query 3:")
print(query3)
cursor_dash.execute(query3)
results3 = cursor_dash.fetchall()
print("Query 3 Results:")
for row in results3:
    std_id, std_name, total_items_bought_price = row
    print(f"Student ID: {std_id}, Student Name: {std_name}, Total Items Bought Price: ${total_items_bought_price}")
print("\n")

# Close the cursor and connection
cursor_dash.close()

# Establish a connection for SQLAlchemy engine
engine = create_engine(f"mysql+mysqlconnector://{user}:{password}@{host}/{database}")
# Load necessary views into Pandas DataFrames
```

```
sales_df = pd.read_sql('SELECT * FROM sale_info_view', engine)
user_df = pd.read_sql('SELECT * FROM user', engine)
student_df = pd.read_sql('SELECT * FROM student', engine)
uni_df = pd.read_sql('SELECT * FROM university', engine)
student_program_department_university_df = pd.read_sql('SELECT * FROM
student_program_department_university_view', engine)
# Fetch alum data from the database
alum_df = pd.read_sql('SELECT * FROM alum', engine)

# Check the columns in the DataFrame
print(student_program_department_university_df.columns)

# Assuming you have a table named 'mentor_mentee' in your database
mentor_mentee_df = pd.read_sql('SELECT * FROM mentorship', engine)

print("Columns before merging mentor_mentee_df:")
print(mentor_mentee_df.columns)

# Merge mentor_mentee_df with student_df to get university information for mentors
mentor_mentee_df = pd.merge(
   mentor_mentee_df,
   student_df[['Std_id', 'Uni_Id']],
   left_on='Std_id',
   right_on='Std_id',
   how='left'
)

# Print columns after merging
print("Columns after merging mentor_mentee_df:")
print(mentor_mentee_df.columns)

# Merge mentor_mentee_df with university_df to get university names for mentors
mentor_mentee_df = pd.merge(
   mentor_mentee_df,
   uni_df[['Uni_id', 'name']],
   left_on='Uni_Id',
   right_on='Uni_id',
   how='left',
   suffixes=('_mentee', '_mentor')
)

# Print columns after the second merge
print("Columns after the second merge in mentor_mentee_df:")
print(mentor_mentee_df.columns)

# Assuming the columns exist, proceed with clustering
# Use existing columns ('program_name', 'department_name', 'university_name') for clustering
cluster_features = ['program_name', 'department_name', 'university_name']

# One-hot encode categorical variables using pandas get_dummies
features_encoded = pd.get_dummies(student_program_department_university_df[cluster_features],
drop_first=True)
```

```
# Standardize the features
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features_encoded)

# Choose the number of clusters (you can adjust this based on your data)
n_clusters = 3

# Apply K-means clustering
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
student_program_department_university_df['cluster'] = kmeans.fit_predict(features_scaled)
# Group alum data by 'batch' and calculate the average years of experience for each batch
avg_experience_by_batch = alum_df.groupby('batch')['years_of_experience'].mean().reset_index()

# Merge DataFrames to get university_name in sale_info_view
merged_df = pd.merge(sales_df, user_df[['email_id', 'User_id']], left_on='seller_email', right_on='email_id')
merged_df = pd.merge(merged_df, student_df[['user_id', 'Std_id', 'Uni_Id']], left_on='User_id', right_on='user_id')
merged_df = pd.merge(merged_df, uni_df[['Uni_id', 'name']], left_on='Uni_Id', right_on='Uni_id')

# Calculate average listed price and average bought price by university
avg_listed_price = merged_df.groupby('name')['item_listed_price'].mean().reset_index()
avg_bought_price = merged_df.groupby('name')['item_bought_price'].mean().reset_index()

# Merge DataFrames on university_name
avg_prices_df = pd.merge(avg_listed_price, avg_bought_price, on='name', suffixes=('_listed', '_bought'))

# Calculate the difference between average listed price and average bought price
avg_prices_df['price_difference'] = avg_prices_df['item_bought_price'] - avg_prices_df['item_listed_price']

# Create a DataFrame to store calculated profits for each university
universities_profit_data = []

# Iterate through each university and calculate profit using the function
for university_name in student_program_department_university_df['university_name'].unique():
    # Get the university_id from the university table based on the university_name
    uni_id = uni_df.loc[uni_df['name'] == university_name, 'Uni_id'].iloc[0]

    # Call the calculateSalesStatsForUniversity function with the uni_id
    profit_stats = pd.read_sql(f"SELECT calculateSalesStatsForUniversity({uni_id}) AS result", engine)

    # Parse the result string to extract profit, highest sale, etc.
    # Assuming the result string is formatted like: "Total Items Sold: ..., Total Price Obtained: ..., Total Profit: ...,
Highest Sale: ..."
    # You may need to adjust this based on the actual format of the result string
    profit_info = profit_stats['result'].str.extract(r'Total Items Sold: (\d+), Total Price Obtained: (\d+\.\d+), Total Profit:
(\d+\.\d+), Highest Sale: (\d+\.\d+)')

    # Check if extraction was successful and the DataFrame has the expected structure
    if profit_info.shape[1] == 4:
        # Replace NaN values with a default value
        profit_info = profit_info.fillna(0)

        # Create a dictionary with the extracted values
        university_profit = {
```

```
            'university_name': university_name,
            'total_items_sold': int(profit_info.iloc[0, 0]),
            'total_price_obtained': float(profit_info.iloc[0, 1]),
            'total_profit': float(profit_info.iloc[0, 2]),
            'highest_sale': float(profit_info.iloc[0, 3]),
        }

        # Append the dictionary to the list
        universities_profit_data.append(university_profit)
    else:
        print(f"Warning: Unable to extract data for university_name {university_name}")

# Create a DataFrame from the list of dictionaries
universities_by_profit_df = pd.DataFrame(universities_profit_data)

print(student_program_department_university_df['cluster'].unique())

# Initialize Dash app
app = dash.Dash(__name__)

# Define app layout
app.layout = html.Div([
    dcc.Tabs([
        dcc.Tab(label='Prices and Alumns', children=[
            # Dropdown for selecting the metric to display
            dcc.Dropdown(
                id='metric-dropdown',
                options=[
                    {'label': 'Average Listed Price', 'value': 'item_listed_price'},
                    {'label': 'Average Bought Price', 'value': 'item_bought_price'},
                    {'label': 'Price Difference', 'value': 'price_difference'},
                ],
                value='price_difference',
                placeholder='Select Metric'
            ),

            # Graph for displaying the selected metric
            dcc.Graph(id='avg-prices-graph'),

            dcc.Input(id='some-input-element', style={'display': 'none'}),
            # Add a line graph
            dcc.Graph(id='line-graph'),

              # Add a new graph for displaying mentorship statistics
            dcc.Graph(id='mentorship-graph'),  # Add this line
        ]),
        dcc.Tab(label='University Stats', children=[
            # Dropdown for Department filter
            dcc.Dropdown(
                id='department-dropdown',
                options=[
                    {'label': dep, 'value': dep}
                    for dep in student_program_department_university_df['department_name'].unique()
```

```
            ],
            value='All',
            multi=True,
            placeholder='Select Department'
        ),

        # Bar chart for Students by Department
        dcc.Graph(id='bar-chart'),

        dcc.Dropdown(
            id='cluster-dropdown',
            options=[
                {'label': f'Cluster {i}', 'value': i} for i in range(n_clusters)
            ] + [{'label': 'All Clusters', 'value': 'All'}],
            value='All',
            placeholder='Select Cluster'
        ),

        # 3D Scatter plot for clustering
        dcc.Graph(id='cluster-plot'),

        # Dropdown for selecting Pie Chart type
        dcc.Dropdown(
            id='pie-chart-type',
            options=[
                {'label': 'Students by University', 'value': 'university'},
                {'label': 'Students by Program', 'value': 'program'}
            ],
            value='university',
            placeholder='Select Pie Chart Type'
        ),

        # Pie chart for Students by University or Program
        dcc.Graph(id='pie-chart'),

        # Dropdown for University filter
        dcc.Dropdown(
            id='university-dropdown',
            options=[
                {'label': uni, 'value': uni}
                for uni in universities_by_profit_df['university_name'].unique()
            ],
            value='All',
            multi=True,
            placeholder='Select University'
        ),

        # Bar chart for Total Profit by University
        dcc.Graph(id='profit-bar-chart'),
    ]),
  ]),
])
```

```
# Define callback to update the graph based on the selected metric
@app.callback(
    Output('avg-prices-graph', 'figure'),
    [Input('metric-dropdown', 'value')]
)
def update_avg_prices_graph(selected_metric):
    # Reverse the values for the Price Difference metric
    if selected_metric == 'price_difference':
        avg_prices_df['price_difference'] = -avg_prices_df['price_difference']

    # Create the graph based on the selected metric
    fig = px.bar(
        avg_prices_df,
        x='name',
        y=selected_metric,
        title=f'{selected_metric.capitalize()} by University',
        labels={'value': 'Price', 'variable': 'Metric'},
    )
    return fig

# Define callback to update bar chart based on department filter
@app.callback(
    Output('bar-chart', 'figure'),
    [Input('department-dropdown', 'value')]
)
def update_chart(selected_departments):
    if not selected_departments or 'All' in selected_departments:
        filtered_df = student_program_department_university_df
    else:
        filtered_df = student_program_department_university_df[
            student_program_department_university_df['department_name'].isin(
                selected_departments
            )
        ]

    # Create bar chart
    fig = px.bar(
        filtered_df,
        x='department_name',
        y='student_id',
        title='Students by Department',
    )
    return fig

# Define callback to update pie chart for Students by University or Program
@app.callback(
    Output('pie-chart', 'figure'),
    [Input('department-dropdown', 'value'),
     Input('pie-chart-type', 'value')]
)
def update_pie_chart(selected_departments, pie_chart_type):
    if not selected_departments or 'All' in selected_departments:
        filtered_df = student_program_department_university_df
```

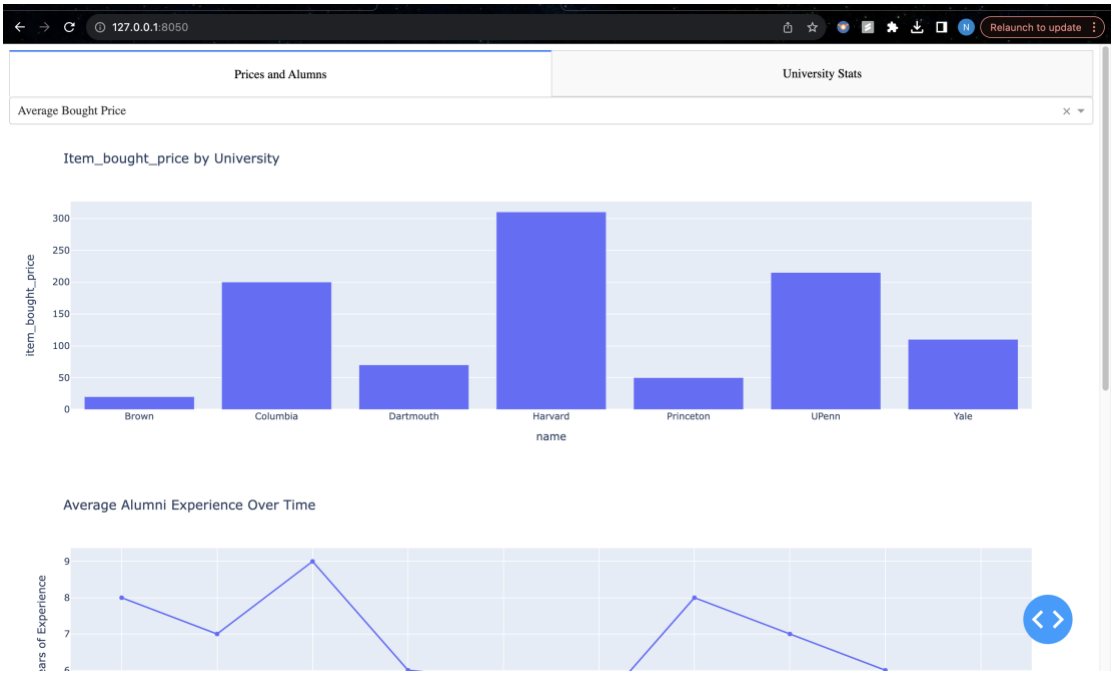
```
    else:
        filtered_df = student_program_department_university_df[
            student_program_department_university_df['department_name'].isin(
                selected_departments
            )
        ]

    # Create pie chart
    if pie_chart_type == 'university':
        fig = px.pie(
            filtered_df, names='university_name', title='Students by University'
        )
    else:
        fig = px.pie(
            filtered_df, names='program_name', title='Students by Program'
        )

    return fig

# Define callback to update improved 3D Scatter plot based on cluster selection
@app.callback(
    [Output('cluster-plot', 'figure')],
    [Input('cluster-dropdown', 'value')]
)
def update_cluster_plot(selected_cluster):
    if selected_cluster == 'All':
        filtered_df = student_program_department_university_df
    else:
        filtered_df = student_program_department_university_df[
            student_program_department_university_df['cluster'] == selected_cluster
        ]

    # Create improved 3D Scatter plot
    fig = px.scatter_3d(
        filtered_df,
        x='program_name',
        y='department_name',
        z='university_name',
        color='cluster',
        title='Improved 3D Scatter Plot for Clustering',
        size_max=10,  # Adjust the maximum size of data points
        opacity=0.7,  # Set the opacity of data points
        symbol='cluster',  # Differentiate clusters with symbols
        symbol_sequence=['circle', 'square', 'diamond'],  # Define symbols for clusters
        height=600,  # Set the height of the plot
        width=800,  # Set the width of the plot
    )

    fig.update_layout(
        scene=dict(
            aspectmode="manual",
            aspectratio=dict(x=2, y=2, z=0.5)  # Adjust the aspect ratio for a better view
        ),
```

```
        legend=dict(
            title='Cluster',
            traceorder='normal',
            orientation='h',  # Set the orientation of the legend to horizontal
            y=1.02,  # Adjust the y position of the legend
            x=0.5  # Center the legend horizontally
        )
    )

    return [fig]

# Define callback to update the line graph based on the selected value
@app.callback(
    Output('line-graph', 'figure'),
    [Input('some-input-element', 'value')]
)
def update_line_graph(selected_value):
    # Group alum data by 'batch' and calculate the average years of experience for each batch
    avg_experience_by_batch = alum_df.groupby('batch')['years_of_experience'].mean().reset_index()

    # Create the line graph based on the average years of experience
    fig = px.line(
        avg_experience_by_batch,
        x='batch',
        y='years_of_experience',
        title='Average Alumni Experience Over Time',
        labels={'years_of_experience': 'Average Years of Experience', 'batch': 'Batch'},
        markers=True
    )

    return fig

# Define callback to update mentorship graph based on university filter
@app.callback(
    Output('mentorship-graph', 'figure'),
    [Input('university-dropdown', 'value')]
)
def update_mentorship_chart(selected_university):
    if not selected_university or 'All' in selected_university:
        filtered_df = mentor_mentee_df
    else:
        filtered_df = mentor_mentee_df[
            (mentor_mentee_df['Uni_Id'].isin(selected_university)) |
            (mentor_mentee_df['Uni_id'].isin(selected_university))
        ]

    # Count the number of mentorships for each university
    mentorship_count = filtered_df.groupby('Uni_Id').size().reset_index(name='mentorship_count')

    # Merge mentorship_count with uni_df to get university names for mentors
    mentorship_count = pd.merge(
        mentorship_count,
        uni_df[['Uni_id', 'name']],
```

```
        left_on='Uni_Id',
        right_on='Uni_id',
        how='left'
    )

    # Create bar chart for mentorships by university
    fig = px.bar(
        mentorship_count,
        x='mentorship_count',
        y='name',  # Use 'name' instead of 'Uni_Id'
        orientation='h',  # Set orientation to horizontal for bar chart
        title='Mentorship Count by University',
        labels={'mentorship_count': 'Number of Mentorships', 'name': 'University Name'},
    )
    return fig

# Define callback to update bar chart based on university filter
@app.callback(
    Output('profit-bar-chart', 'figure'),
    [Input('university-dropdown', 'value')]
)
def update_profit_chart(selected_universities):
    if not selected_universities or 'All' in selected_universities:
        filtered_df = universities_by_profit_df
    else:
        filtered_df = universities_by_profit_df[
            universities_by_profit_df['university_name'].isin(selected_universities)
        ]

    # Create bar chart for total profit by university
    fig = px.bar(
        filtered_df,
        x='university_name',
        y='total_profit',
        title='Total Profit by University',
    )
    return fig

# Run the app
if __name__ == '__main__':
    app.run_server(debug=True)
```

ANALYSIS OVERVIEW:

Page 1



Page 2