UNIVERSITY OF MICHIGAN

Week 4: Dashboard

For this notebook, you are building a prototype dashboard. The specific user persona we have in mind for you to design around is a non-technical instructor who is relatively new to teaching online and has moderate to low data literacy skills.

The instructor, as we have mentioned in previous Notebooks, is looking to run a report about 25% of the way through the course in order to identify who he or she should do a formal check-in with.

To help you in building your dashboard, this notebook provides a brief introduction to the jupyter-dash extension for building a Plotly Dash app within Jupyter environments. You can check out the documentation and some tutorials for Plotly Dash here.

Resources:

- A medium blog - Introducing JupyterDash
- Plotly Dash documentation
- A Youtube tutorial on dash
- Dash gallery

You can also check out OU Analyse as an example of how the authors of the dataset have developed a dashboard for their institution. You can request a demo of the dashboard by entering your email.

In [2]:
```
# !pip install jupyter-dash  ## should already be installed

import pandas as pd
import numpy as np

# Import jupyter dash
from jupyter_dash import JupyterDash
import os
try:
    os.environ.pop('http_proxy')
    os.environ.pop('https_proxy')
except KeyError:
    pass

# Import dash
import dash
from dash import dcc
from dash import html

# Import plotly
import plotly.graph_objs as go
import plotly.express as px

# Set up jupyter proxy
JupyterDash.infer_jupyter_proxy_config()
```

In [3]:
```
lr = pd.read_csv('assets/learning_resources.csv')
qt = pd.read_csv('assets/quizzes_tests.csv')
si = pd.read_csv('assets/student_info_pred.csv')
df = pd.read_csv('assets/country_indicators.csv')
```

A quick tutorial

In essence, a plotly-dash dashboard consists of 3 components:

- The **dash components** (e.g., dropdown, slider, checklist, etc.). See the documentation here
- The **plotly** graphs (e.g., linegraph, scatter plot, heatmap, etc.). See the documentation here
- The **callback** to connects the dash components to plotly graphs, making it an interactive dashboard. See the documentation here

Step 1. The Dash components (i.e. layouts)

In [4]:
```
score = qt.groupby('assignment_name')['score'].agg({np.mean,np.median, np.std}).reset_index(drop = False)
score.head()
```

Out[4]:

	assignment_name	std	median	mean
0	Final Exam	28.804439	47.0	41.728167
1	Quiz 1	22.864709	70.0	66.568266
2	Quiz 2	27.877683	83.0	75.221402
3	Quiz 3	30.384604	72.0	63.538745
4	Quiz 4	32.361511	72.0	61.413284

In [5]:
```
app = JupyterDash(__name__)

# Create server variable
server = app.server

# Create a unique list of code_module
available_indicators1 = score['assignment_name'].unique()

# Step 1

app.layout = html.Div([

            # Create a html title for the dashboard
            html.H1("This is the title"),

            # Create a graph, we will configure the graph using plotly express in step 3
            dcc.Graph(id='graph-with-dropdown'),

            # Create a dropdown menu based on code_module
            dcc.Dropdown(
```

```
                id='crossfilter-xaxis-column',
                options=[{'label': i, 'value': i} for i in available_indicators1],
                value='Quiz 1' # the default assignment
            )])

# Run the app
app.run_server(mode="inline", port = 8100)

# You will see we have the dropdown menu but nothing happens yet
```

Step 2 & 3. Callback and create plotly graph

In [6]:
```
# Create a unique list of code_module
assess_list = score['assignment_name'].unique()

# Step 1
app.layout = html.Div([
            # Create a html title for the dashboard
            html.H1("This is a bar chart"),

            # Create a graph, we will configure the graph using plotly express in step 3
            dcc.Graph(id='graph-with-dropdown'),

            # Create a dropdown menu based on code_module
            dcc.Dropdown(
                id='crossfilter-xaxis-column',
                options=[{'label': i, 'value': i} for i in assess_list],
                value='Quiz 1' # the default
            )])

# Step 2
# Callback using input from dropdown menu to generate graph
# You can have multiple inputs and multiple outputs
@app.callback(
    dash.dependencies.Output('graph-with-dropdown', 'figure'),
    [dash.dependencies.Input('crossfilter-xaxis-column', 'value')])

# Step 3
# Define the graph with plotly express
def update_figure(assignment_name):
    filtered_score = score[(score.assignment_name == assignment_name)]
    figure = px.bar(filtered_score, x = 'assignment_name', y = 'mean')
    return figure # You must return all the output(s) in step 2

# Run the app
app.run_server(mode = "inline", port = 8101)
```

In [7]:
```
assess_list
```

Out[7]:
```
array(['Final Exam', 'Quiz 1', 'Quiz 2', 'Quiz 3', 'Quiz 4', 'Quiz 5',
       'Quiz 6', 'Quiz 7', 'Test 1', 'Test 2', 'Test 3', 'Test 4',
       'Test 5', 'Test 6'], dtype=object)
```

In [8]:
```
# Create a unique list of assignment_name
assess_list = score['assignment_name'].unique()

# Step 1
app.layout = html.Div([
            # Create a html title for the dashboard
            html.H1("This is a scatter plot"),

            # Create a graph, we will configure the graph using plotly express in step 3
            dcc.Graph(id='graph'),

            # Create a checklist based on assignment_name
            dcc.Checklist(
                    id = 'checklist',
                    options=[{'label': i, 'value': i} for i in assess_list],
                    value=assess_list # Default values contain all assignment_name
            )
])

# Step 2
# Callback using inputs from the checklist to generate the graph
@app.callback(
    dash.dependencies.Output('graph', 'figure'),
    [dash.dependencies.Input('checklist', 'value')])

# Step 3
# Define the graph with plotly express
def update_figure(assignment_name):
    figure = px.scatter(score,
                     x="mean",
                     y="std",
                     color="assignment_name",
                     hover_name='assignment_name')
    return figure # You must return all the output(s) in step 2

# Run the app
app.run_server(mode="inline", port = 8102)
```

Building Your Dashboard (40 points)

Your final product will need to have the following capabilities:

1. (10 points) View multiple students' predicted probabilities of failing the course
2. (10 points) View a student's quiz/test performances alongside a meaningful reference like a course average for a given quiz/test
3. (10 points) View a student's learning resource use alongside a meaningful reference like a course average for a given resource
4. (10 points) Intergrate the above three capabilties into a single, functional dashboard

This notebook is broken out into four sections, three to build individual components and one to integrate them. The final Integrated Dashboard cell will be graded and must include all three specified capabilities. If it does not, points will be awarded based on progress made in each Component cell.

There is a new column in `student_info.csv`, `fail_pred,` that represents the predicted probability of failure for each student from a baseline model.

Note: All quizzes, tests, and resources should only include information from on or before day 60 in the course.

Dictionary

- **student_info.csv**
  - **id_student** = numeric; unique identifier for each student in the course
  - **gender** = character; M = "male", F = "female"

- **highest_education** = character; "Some Graduate", "Some Higher Education", "High School + Advanced Placement", "High School", "No Formal Quals" (Categories ordered from highest documented education level attained to lowest documented education level attained)
- **disability** = character; Y = "yes", N = "no"
- **final_result** = character; "Fail", "Pass"
- **fail_pred** = numeric; predicted probability from sample model

- **quizzes_tests.csv**
  - **id_student** = numeric; unique identifier for each student in the course
  - **assignment_name** = character; name of graded assignment (Quiz 1-7, Test 1-6, Final Exam)
  - **due_date** = numeric; date assignment was due (indexed as count in days from start of course, i.e., day 0)
  - **weight** = numeric; weight multiplied by score when generating final grade (weight * score / 100)
  - **date_submitted** = numeric; date student submitted assigned (indexed as count in days from start of course, i.e., day 0, NaN means students did not submit assignment)
  - **score** = numeric; score student earned on assignment (0 means students did not submit assignment)

- **learning_resources.csv**
  - **id_student** = numeric; unique identifier for each student in the course
  - **activity_type** = character; overarching label for learning activity students can access ("course_homepage", "course_page", "forum", 'resource", "wiki")
  - **activity_id** = numeric; unique identifier for specific learning activity student accessed within overacting `activity_type`
  - **date** = numeric; date student accessed specific `acitivity_id` (indexed as count in days from start of course, i.e., day 0)
  - **sum_click** = numeric; count of clicks for `activity_id` on date

Component 1

1. (10 points) View multiple students' predicted probabilities of failing the course

In [9]:
```
si.head()
```

Out[9]:

	pred_fail	id_student	gender	highest_education	disability	final_result
0	0.133394	41060	M	Some Higher Education	N	Fail
1	0.869518	45664	M	Some Higher Education	N	Pass
2	0.472989	52014	F	High School	N	Fail
3	0.213423	53488	F	Some Higher Education	N	Pass
4	0.290116	60135	F	High School + Advanced Placement	N	Pass

In [10]:
```
si.shape
```

Out[10]: (813, 6)

In [11]:
```
fail_preds = si[['pred_fail', 'id_student']]
fail_preds = fail_preds.drop(812)
fail_preds.head()
```

Out[11]:

	pred_fail	id_student
0	0.133394	41060
1	0.869518	45664
2	0.472989	52014
3	0.213423	53488
4	0.290116	60135

In [12]:
```
fail_preds.shape
```

Out[12]: (812, 2)

In [13]:
```
app = JupyterDash(__name__)
server = app.server


# Step 1: Create a layout (title, dropdown menu, slider, etc...)

# Step 2: Callback to connect input(s) to output(s)

# Step 3: Define the graph with plotly express


# Un-comment to run the app
# app.run_server(mode="inline", port = 8052)
```

Component 2

2. (10 points) View a student's quiz/test performances alongside a meaningful reference like a course average for a given quiz/test

Note: All quizzes, tests, and resources should be from on or before day 60 in the course.

In [14]:
```
# YOUR CODE HERE
qt.head()
```

Out[14]:

	id_student	assignment_name	due_date	weight	date_submitted	score
0	41060	Quiz 1	23	2.0	25.0	77
1	41060	Test 1	25	7.5	24.0	85
2	41060	Quiz 2	51	3.0	54.0	94
3	41060	Test 2	53	10.0	53.0	86
4	41060	Quiz 3	79	3.0	81.0	94

In [15]:
```
qt.shape
```

Out[15]: (11382, 6)

In [16]:
```
qt['weight-score'] = (qt['weight']*qt['score'])/100
qt.head()
```

Out[16]:

	id_student	assignment_name	due_date	weight	date_submitted	score	weight-score
0	41060	Quiz 1	23	2.0	25.0	77	1.540
1	41060	Test 1	25	7.5	24.0	85	6.375
2	41060	Quiz 2	51	3.0	54.0	94	2.820
3	41060	Test 2	53	10.0	53.0	86	8.600
4	41060	Quiz 3	79	3.0	81.0	94	2.820

In [17]:
```
qt_new = qt.loc[(qt['due_date']<=60) & (qt['date_submitted']<=60)]
qt_new.head()
```

Out[17]:

	id_student	assignment_name	due_date	weight	date_submitted	score	weight-score
0	41060	Quiz 1	23	2.0	25.0	77	1.540
1	41060	Test 1	25	7.5	24.0	85	6.375
2	41060	Quiz 2	51	3.0	54.0	94	2.820
3	41060	Test 2	53	10.0	53.0	86	8.600
14	45664	Quiz 1	23	2.0	25.0	47	0.940

In [18]:
```
qt_new.shape
```

Out[18]: (3013, 7)

In [19]:
```
score = qt.groupby('assignment_name')['score'].agg({np.mean,np.median, np.std}).reset_index(drop = False)
score.head()
```

Out[19]:

	assignment_name	std	median	mean
0	Final Exam	28.804439	47.0	41.728167
1	Quiz 1	22.864709	70.0	66.568266
2	Quiz 2	27.877683	83.0	75.221402
3	Quiz 3	30.384604	72.0	63.538745
4	Quiz 4	32.361511	72.0	61.413284

In [20]:
```
score.shape
```

Out[20]: (14, 4)

In [21]:
```
qt_new1 = pd.pivot_table(qt_new, index='id_student', columns='assignment_name', values='weight-score', aggfunc={'weight-score
qt_new1.head()
```

Out[21]:

assignment_name	id_student	Quiz 1	Quiz 2	Test 1	Test 2
0	41060	1.54	2.82	6.375	8.6
1	45664	0.94	0	4.95	4.6
2	52014	1.06	1.59	6.075	7.0
3	53488	1.86	2.91	6.0	6.1
4	60135	1.14	2.58	5.7	7.6

In [22]:
```
qt_new1.reset_index
qt_new1.head()
```

Out[22]:

assignment_name	id_student	Quiz 1	Quiz 2	Test 1	Test 2
0	41060	1.54	2.82	6.375	8.6
1	45664	0.94	0	4.95	4.6
2	52014	1.06	1.59	6.075	7.0
3	53488	1.86	2.91	6.0	6.1
4	60135	1.14	2.58	5.7	7.6

In [23]:
```
qt_new1.shape
```

Out[23]: (812, 5)

In [24]:
```
qt_new1['Quiz 1'].dtype
```

Out[24]: dtype('O')

In [25]:
```
qt_new1['Quiz 1'] = qt_new1['Quiz 1'].astype(float)
qt_new1['Quiz 2'] = qt_new1['Quiz 2'].astype(float)
qt_new1['Test 1'] = qt_new1['Test 1'].astype(float)
qt_new1['Test 2'] = qt_new1['Test 2'].astype(float)
qt_new1['Quiz 1'].dtype
```

Out[25]: dtype('float64')

In [26]:
```
Quiz_1_mean = qt_new1['Quiz 1'].mean()
```

In [27]:
```
Quiz_2_mean = qt_new1['Quiz 2'].mean()
```

In [28]:
```
Test_1_mean = qt_new1['Test 1'].mean()
```

In [29]:
```
Test_2_mean = qt_new1['Test 2'].mean()
```

In [30]:
```
qt_new1['Quiz 1 Average'] = Quiz_1_mean
qt_new1['Quiz 2 Average'] = Quiz_2_mean
qt_new1['Test 1 Average'] = Test_1_mean
qt_new1['Test 2 Average'] = Test_2_mean
qt_new1.head()
```

Out[30]:

assignment_name	id_student	Quiz 1	Quiz 2	Test 1	Test 2	Quiz 1 Average	Quiz 2 Average	Test 1 Average	Test 2 Average
0	41060	1.54	2.82	6.375	8.6	1.333005	2.256872	5.337931	6.443719
1	45664	0.94	0.00	4.950	4.6	1.333005	2.256872	5.337931	6.443719
2	52014	1.06	1.59	6.075	7.0	1.333005	2.256872	5.337931	6.443719
3	53488	1.86	2.91	6.000	6.1	1.333005	2.256872	5.337931	6.443719
4	60135	1.14	2.58	5.700	7.6	1.333005	2.256872	5.337931	6.443719

In [ ]:
```

```

In [31]:
```
qt_new1['Quiz 1 percent compared to average'] = (qt_new1['Quiz 1']/qt_new1['Quiz 1 Average'])*100
qt_new1['Quiz 1 percent compared to average'] = round(qt_new1['Quiz 1 percent compared to average'], 0)
qt_new1.head()
```

Out[31]:

assignment_name	id_student	Quiz 1	Quiz 2	Test 1	Test 2	Quiz 1 Average	Quiz 2 Average	Test 1 Average	Test 2 Average	Quiz 1 percent compared to average
0	41060	1.54	2.82	6.375	8.6	1.333005	2.256872	5.337931	6.443719	116.0
1	45664	0.94	0.00	4.950	4.6	1.333005	2.256872	5.337931	6.443719	71.0
2	52014	1.06	1.59	6.075	7.0	1.333005	2.256872	5.337931	6.443719	80.0
3	53488	1.86	2.91	6.000	6.1	1.333005	2.256872	5.337931	6.443719	140.0
4	60135	1.14	2.58	5.700	7.6	1.333005	2.256872	5.337931	6.443719	86.0

In [32]:
```
less_than_avg_Quiz1 = qt_new1[qt_new1['Quiz 1'] < 1.333005]
```

less_than_avg_Quiz1

Out[32]:

assignment_name	id_student	Quiz 1	Quiz 2	Test 1	Test 2	Quiz 1 Average	Quiz 2 Average	Test 1 Average	Test 2 Average	Quiz 1 percent compared to average
1	45664	0.94	0.00	4.950	4.6	1.333005	2.256872	5.337931	6.443719	71.0
2	52014	1.06	1.59	6.075	7.0	1.333005	2.256872	5.337931	6.443719	80.0
4	60135	1.14	2.58	5.700	7.6	1.333005	2.256872	5.337931	6.443719	86.0
6	67602	0.86	2.01	4.050	5.2	1.333005	2.256872	5.337931	6.443719	65.0
8	69491	1.20	2.67	5.925	4.8	1.333005	2.256872	5.337931	6.443719	90.0
...	...	...	...	...	...	...	...	...	...	...
801	2659756	1.00	0.00	3.975	0.0	1.333005	2.256872	5.337931	6.443719	75.0
802	2661870	1.06	1.92	5.475	5.0	1.333005	2.256872	5.337931	6.443719	80.0
804	2669206	0.94	2.76	3.300	0.0	1.333005	2.256872	5.337931	6.443719	71.0
807	2678338	1.00	2.67	5.625	7.2	1.333005	2.256872	5.337931	6.443719	75.0
811	2694933	1.14	2.76	6.825	8.9	1.333005	2.256872	5.337931	6.443719	86.0

326 rows × 10 columns

Component 3

3. (10 points) View a student's learning resource use alongside a meaningful reference like a course average for a given resource

Note: All quizzes, tests, and resources should be from on or before day 60 in the course.

In [33]:
```
# YOUR CODE HERE
lr.head()
```

Out[33]:

	id_student	activity_type	activity_id	date	sum_click
0	420388	resource	219	0	1
1	420388	course_homepage	1	0	3
2	420388	course_page	87	0	1
3	420388	resource	229	0	2
4	420388	course_page	1	0	2

In [34]:
```
lr.shape
```

Out[34]: (443336, 5)

In [35]:
```
lr_sorted = lr.loc[(lr['date']<=60)]
```

In [36]:
```
lr_wide_sum = pd.pivot_table(lr_sorted, index='id_student', columns='activity_type', values='sum_click', aggfunc='sum').filln
lr_wide_sum.head()
```

Out[36]:

activity_type	id_student	course_homepage	course_page	forum	resource	wiki
0	41060	37.0	108.0	16.0	19.0	11.0
1	45664	144.0	141.0	107.0	115.0	29.0
2	52014	47.0	43.0	26.0	8.0	20.0
3	53488	173.0	98.0	332.0	65.0	39.0
4	60135	629.0	456.0	456.0	185.0	78.0

In [37]:
```
lr_wide_sum['course_homepage'].dtype
```

Out[37]: dtype('O')

In [38]:
```
lr_wide_sum['course_homepage'] = lr_wide_sum['course_homepage'].astype(float)
lr_wide_sum['course_page'] = lr_wide_sum['course_page'].astype(float)
lr_wide_sum['forum'] = lr_wide_sum['forum'].astype(float)
lr_wide_sum['resource'] = lr_wide_sum['resource'].astype(float)
lr_wide_sum['wiki'] = lr_wide_sum['wiki'].astype(float)
lr_wide_sum['forum'].dtype
```

Out[38]: dtype('float64')

In [39]:
```
course_homepage_mean = lr_wide_sum['course_homepage'].mean()
course_page_mean = lr_wide_sum['course_page'].mean()
forum_mean = lr_wide_sum['forum'].mean()
resource_mean = lr_wide_sum['resource'].mean()
wiki_mean = lr_wide_sum['wiki'].mean()
```

In [40]:
```
course_homepage_std = lr_wide_sum['course_homepage'].std()
course_page_std = lr_wide_sum['course_page'].std()
forum_std = lr_wide_sum['forum'].std()
resource_std = lr_wide_sum['resource'].std()
wiki_std = lr_wide_sum['wiki'].std()
```

In [41]:
```
course_homepage_sum = lr_wide_sum['course_homepage'].sum()
course_page_sum = lr_wide_sum['course_page'].sum()
forum_sum = lr_wide_sum['forum'].sum()
resource_sum = lr_wide_sum['resource'].sum()
wiki_sum = lr_wide_sum['wiki'].sum()
```

In [42]:
```
lr_wide_sum['course_homepage_average'] = course_homepage_mean
lr_wide_sum['course_page_average'] = course_page_mean
lr_wide_sum['forum_average'] = forum_mean
lr_wide_sum['resource_average'] = resource_mean
lr_wide_sum['wiki_average'] = wiki_mean

lr_wide_sum['course_homepage_std'] = course_homepage_std
lr_wide_sum['course_page_std'] = course_page_std
lr_wide_sum['forum_std'] = forum_std
lr_wide_sum['resource_std'] = resource_std
lr_wide_sum['wiki_std'] = wiki_std

lr_wide_sum['course_homepage_sum'] = course_homepage_sum
lr_wide_sum['course_page_sum'] = course_page_sum
lr_wide_sum['forum_sum'] = forum_sum
lr_wide_sum['resource_sum'] = resource_sum
lr_wide_sum['wiki_sum'] = wiki_sum

lr_wide_sum.head()
```

Out[42]:

activity_type	id_student	course_homepage	course_page	forum	resource	wiki	course_homepage_average	course_page_average	forum_average	resourc
0	41060	37.0	108.0	16.0	19.0	11.0	178.428571	192.461823	152.160099	
1	45664	144.0	141.0	107.0	115.0	29.0	178.428571	192.461823	152.160099	

2	52014	47.0	43.0	26.0	8.0	20.0	178.428571	192.461823	152.160099	
3	53488	173.0	98.0	332.0	65.0	39.0	178.428571	192.461823	152.160099	
4	60135	629.0	456.0	456.0	185.0	78.0	178.428571	192.461823	152.160099	

5 rows × 21 columns

In [43]:
```
lr_wide_sum.shape
```

Out[43]: (812, 21)

In [44]:
```
#lr_wide_sum['rate of utlizing resources'] = lr_wide_sum['resource']/lr_wide_sum['resource_sum']
lr_wide_sum['freq_of_resource_use_compared_to_avg'] = np.where(lr_wide_sum['resource']<=78.498768, 0, 1)
lr_wide_sum.head(20)
```

Out[44]:

activity_type	id_student	course_homepage	course_page	forum	resource	wiki	course_homepage_average	course_page_average	forum_average	resourc
0	41060	37.0	108.0	16.0	19.0	11.0	178.428571	192.461823	152.160099	
1	45664	144.0	141.0	107.0	115.0	29.0	178.428571	192.461823	152.160099	
2	52014	47.0	43.0	26.0	8.0	20.0	178.428571	192.461823	152.160099	
3	53488	173.0	98.0	332.0	65.0	39.0	178.428571	192.461823	152.160099	
4	60135	629.0	456.0	456.0	185.0	78.0	178.428571	192.461823	152.160099	
5	66579	112.0	85.0	165.0	67.0	36.0	178.428571	192.461823	152.160099	
6	67602	80.0	123.0	71.0	44.0	11.0	178.428571	192.461823	152.160099	
7	67785	113.0	132.0	57.0	70.0	9.0	178.428571	192.461823	152.160099	
8	69491	34.0	41.0	3.0	24.0	0.0	178.428571	192.461823	152.160099	
9	76536	9.0	0.0	8.0	1.0	0.0	178.428571	192.461823	152.160099	
10	77763	128.0	166.0	105.0	88.0	30.0	178.428571	192.461823	152.160099	
11	81351	97.0	137.0	44.0	68.0	36.0	178.428571	192.461823	152.160099	
12	86047	69.0	137.0	88.0	43.0	23.0	178.428571	192.461823	152.160099	
13	87107	113.0	69.0	82.0	67.0	42.0	178.428571	192.461823	152.160099	
14	89051	138.0	211.0	95.0	78.0	23.0	178.428571	192.461823	152.160099	
15	92775	3100.0	961.0	715.0	1474.0	84.0	178.428571	192.461823	152.160099	
16	94923	69.0	107.0	92.0	28.0	36.0	178.428571	192.461823	152.160099	
17	94964	27.0	60.0	67.0	16.0	0.0	178.428571	192.461823	152.160099	
18	96592	74.0	108.0	45.0	41.0	14.0	178.428571	192.461823	152.160099	
19	103904	131.0	145.0	84.0	101.0	0.0	178.428571	192.461823	152.160099	

20 rows × 22 columns

Integrated Dashboard

4. (10 points) Intergrate the above three capabilties into a single, functional dashboard

In [45]:
```
data_merged1 = fail_preds.merge(qt_new1, on='id_student', how='left')
data_merged2 = data_merged1.merge(lr_wide_sum, on='id_student', how='left')
data_merged2.head()
```

Out[45]:

	pred_fail	id_student	Quiz 1	Quiz 2	Test 1	Test 2	Quiz 1 Average	Quiz 2 Average	Test 1 Average	Test 2 Average	...	course_page_std	forum_std	resource_std	wiki_std	course
0	0.133394	41060	1.54	2.82	6.375	8.6	1.333005	2.256872	5.337931	6.443719	...	141.758186	156.552482	80.02881	32.187357	
1	0.869518	45664	0.94	0.00	4.950	4.6	1.333005	2.256872	5.337931	6.443719	...	141.758186	156.552482	80.02881	32.187357	
2	0.472989	52014	1.06	1.59	6.075	7.0	1.333005	2.256872	5.337931	6.443719	...	141.758186	156.552482	80.02881	32.187357	
3	0.213423	53488	1.86	2.91	6.000	6.1	1.333005	2.256872	5.337931	6.443719	...	141.758186	156.552482	80.02881	32.187357	
4	0.290116	60135	1.14	2.58	5.700	7.6	1.333005	2.256872	5.337931	6.443719	...	141.758186	156.552482	80.02881	32.187357	

5 rows × 32 columns

In [46]:
```
data_merged2['resource'].reset_index()
```

Out[46]:

	index	resource
0	0	19.0
1	1	115.0
2	2	8.0
3	3	65.0
4	4	185.0
...	...	...
807	807	55.0
808	808	108.0
809	809	47.0
810	810	43.0
811	811	137.0

812 rows × 2 columns

In [47]:
```
data_merged2
```

Out[47]:

	pred_fail	id_student	Quiz 1	Quiz 2	Test 1	Test 2	Quiz 1 Average	Quiz 2 Average	Test 1 Average	Test 2 Average	...	course_page_std	forum_std	resource_std	wiki_std	cou
0	0.133394	41060	1.54	2.82	6.375	8.6	1.333005	2.256872	5.337931	6.443719	...	141.758186	156.552482	80.02881	32.187357	
1	0.869518	45664	0.94	0.00	4.950	4.6	1.333005	2.256872	5.337931	6.443719	...	141.758186	156.552482	80.02881	32.187357	
2	0.472989	52014	1.06	1.59	6.075	7.0	1.333005	2.256872	5.337931	6.443719	...	141.758186	156.552482	80.02881	32.187357	
3	0.213423	53488	1.86	2.91	6.000	6.1	1.333005	2.256872	5.337931	6.443719	...	141.758186	156.552482	80.02881	32.187357	
4	0.290116	60135	1.14	2.58	5.700	7.6	1.333005	2.256872	5.337931	6.443719	...	141.758186	156.552482	80.02881	32.187357	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
807	0.069285	2677955	1.74	2.82	7.125	9.8	1.333005	2.256872	5.337931	6.443719	...	141.758186	156.552482	80.02881	32.187357	
808	0.341921	2678338	1.00	2.67	5.625	7.2	1.333005	2.256872	5.337931	6.443719	...	141.758186	156.552482	80.02881	32.187357	
809	0.167281	2683836	1.60	2.76	6.300	7.8	1.333005	2.256872	5.337931	6.443719	...	141.758186	156.552482	80.02881	32.187357	

810	0.116698	2689536	1.54	3.00	6.750	8.6	1.333005	2.256872	5.337931	6.443719	...	141.758186	156.552482	80.02881	32.187357
811	0.136478	2693243	1.94	2.91	5.775	7.5	1.333005	2.256872	5.337931	6.443719	...	141.758186	156.552482	80.02881	32.187357

812 rows × 32 columns

In [48]:
```
import plotly.express as px
```

In [49]:
```
pred_fail = data_merged2['pred_fail']
Quiz_2 = data_merged2['Quiz 2']
Test_2 = data_merged2['Test 2']
resource = data_merged2['resource']
data_merged3 = data_merged2.rename(columns={"Quiz 2": "Quiz_2", "Test 2": "Test_2", "Quiz 1":"Quiz_1", "Test 1":"Test_1"})
#filtered_df = data_merged3.filter(items=['pred_fail', 'Quiz_2', 'Test_2', 'resource'])
data_merged3.head()
```

Out[49]:

	pred_fail	id_student	Quiz_1	Quiz_2	Test_1	Test_2	Quiz 1 Average	Quiz 2 Average	Test 1 Average	Test 2 Average	...	course_page_std	forum_std	resource_std	wiki_std
0	0.133394	41060	1.54	2.82	6.375	8.6	1.333005	2.256872	5.337931	6.443719	...	141.758186	156.552482	80.02881	32.187357
1	0.869518	45664	0.94	0.00	4.950	4.6	1.333005	2.256872	5.337931	6.443719	...	141.758186	156.552482	80.02881	32.187357
2	0.472989	52014	1.06	1.59	6.075	7.0	1.333005	2.256872	5.337931	6.443719	...	141.758186	156.552482	80.02881	32.187357
3	0.213423	53488	1.86	2.91	6.000	6.1	1.333005	2.256872	5.337931	6.443719	...	141.758186	156.552482	80.02881	32.187357
4	0.290116	60135	1.14	2.58	5.700	7.6	1.333005	2.256872	5.337931	6.443719	...	141.758186	156.552482	80.02881	32.187357

5 rows × 32 columns

In [57]:
```
# App
#2 at a time
app = JupyterDash(__name__)
# Server
server = app.server
# Layout components

unique_student_ids = data_merged3['id_student'].unique()

app.layout = html.Div([
    html.Div([
        html.H1('Prediction of Failure Plot'),
        dcc.Graph(id = 'graph1'),
        dcc.Checklist(
            id='dropdown_id_student1',
            options=[{'label': i, 'value': i} for i in unique_student_ids],
            value=unique_student_ids[:5]
        )
    ]),

    html.Div([
        html.H1("Quiz 2 Plot"),
        dcc.Graph(id = 'graph2'),
        dcc.Checklist(
            id='dropdown_id_student2',
            options=[{'label': i, 'value': i} for i in unique_student_ids],
            value=unique_student_ids
        )
    ]),

    html.Div([
        html.H1("Test 2 Plot"),
        dcc.Graph(id = 'graph3'),
        dcc.Checklist(
            id='dropdown_id_student3',
            options=[{'label': i, 'value': i} for i in unique_student_ids],
            value=unique_student_ids
        )
    ]),

    html.Div([
        html.H1("Resource-Usage Plot"),
        dcc.Graph(id = 'graph4'),
        dcc.Checklist(
            id='dropdown_id_student4',
            options=[{'label': i, 'value': i} for i in unique_student_ids],
            value=unique_student_ids
        )
    ]),
])
# Callbacks
@app.callback(
    dash.dependencies.Output('graph1', 'figure'),
    [dash.dependencies.Input('dropdown_id_student1', 'value')])
def update_figure(id_student):
    figure1 = px.scatter(data_merged3, x = 'id_student', y = 'pred_fail')
    return figure1
@app.callback(
    dash.dependencies.Output('graph2', 'figure'),
    [dash.dependencies.Input('dropdown_id_student2', 'value')])
def update_figure(id_student):
    figure2 = px.histogram(data_merged3, x = 'id_student', y = 'Quiz_2')
    return figure2
@app.callback(
    dash.dependencies.Output('graph3', 'figure'),
    [dash.dependencies.Input('dropdown_id_student3', 'value')])
def update_figure(id_student):
    figure2 = px.histogram(data_merged3, x = 'id_student', y = 'Test_2')
    return figure2
@app.callback(
    dash.dependencies.Output('graph4', 'figure'),
    [dash.dependencies.Input('dropdown_id_student4', 'value')])
def update_figure(id_student):
    figure2 = px.histogram(data_merged3, x = 'id_student', y = 'resource')
    return figure2

# Run
#app.run_server(mode = 'inline', port = 8153)

if __name__ == '__main__':
    app.run_server()
```

Dash app running on https://vcgnaoov.labs.coursera.org/proxy/8050/

In [50]:
```
# Create a unique list of code_module
assess_list = data_merged3['id_student'].unique()

# Step 1
# Layout components
app.layout = html.Div([
```

```
    html.Div([
        html.H1('Prediction of Failure Plot'),
        dcc.Graph(id = 'graph1')]),
    html.Div([
        html.H1("Quiz 1 Plot"),
        dcc.Graph(id = 'graph2')]),
    html.Div([
        html.H1('Quiz 2 Plot'),
        dcc.Graph(id = 'graph3')]),
    # Create a dropdown menu based on code_module
    dcc.Dropdown(
    id='dropdown_id_student',
    options=[{'label': i, 'value': i} for i in assess_list],
    value=assess_list
    )])

# Step 2
# Callback using input from dropdown menu to generate graph
# You can have multiple inputs and multiple outputs
@app.callback(
    dash.dependencies.Output('graph1', 'figure1'),
    dash.dependencies.Output('graph2', 'figure2'),
    dash.dependencies.Output('graph3', 'figure3'),
    [dash.dependencies.Input('dropdown_id_student', 'value')])

# Step 3
# Define the graph with plotly express
def update_figure(id_student):
    filtered_score = data_merged3[(data_merged3.id_student == id_student)]
    figure1 = px.bar(filtered_score, x = 'id_student', y = 'pred_fail')
    return figure1 # You must return all the output(s) in step 2

def update_figure(id_student):
    filtered_score = data_merged3[(data_merged3.id_student == id_student)]
    figure2 = px.bar(filtered_score, x = 'id_student', y = 'Quiz_2')
    return figure2 # You must return all the output(s) in step 2

def update_figure(id_student):
    filtered_score = data_merged3[(data_merged3.id_student == id_student)]
    figure3 = px.bar(filtered_score, x = 'id_student', y = 'Test_2')
    return figure3 # You must return all the output(s) in step 2

# Run the app
#app.run_server(mode = "inline", port = 8101)
if __name__ == '__main__':
    app.run_server()
```

Dash app running on https://vcgnaoov.labs.coursera.org/proxy/8050/

In [51]:
```
data_merged3.columns
```

Out[51]:
```
Index(['pred_fail', 'id_student', 'Quiz_1', 'Quiz_2', 'Test_1', 'Test_2',
       'Quiz 1 Average', 'Quiz 2 Average', 'Test 1 Average', 'Test 2 Average',
       'Quiz 1 percent compared to average', 'course_homepage', 'course_page',
       'forum', 'resource', 'wiki', 'course_homepage_average',
       'course_page_average', 'forum_average', 'resource_average',
       'wiki_average', 'course_homepage_std', 'course_page_std', 'forum_std',
       'resource_std', 'wiki_std', 'course_homepage_sum', 'course_page_sum',
       'forum_sum', 'resource_sum', 'wiki_sum',
       'freq_of_resource_use_compared_to_avg'],
      dtype='object')
```

In [52]:
```
# Create a unique list of assignment_name
assess_list = data_merged3['id_student'].unique()

# Step 1
app.layout = html.Div([
            # Create a html title for the dashboard
            html.H1("This is a scatter plot"),

            # Create a graph, we will configure the graph using plotly express in step 3
            dcc.Graph(id='graph1'),
            dcc.Graph(id='graph2'),
            dcc.Graph(id='graph3'),
            dcc.Graph(id='graph4'),

            # Create a checklist based on assignment_name
            dcc.Checklist(
                    id = 'checklist',
                    options=[{'label': i, 'value': i} for i in assess_list],
                    value=assess_list # Default values contain all assignment_name
            )
])

# Step 2
# Callback using inputs from the checklist to generate the graph
@app.callback(
    dash.dependencies.Output('graph1', 'figure1'),
    dash.dependencies.Output('graph2', 'figure2'),
    dash.dependencies.Output('graph3', 'figure3'),
    dash.dependencies.Output('graph4', 'figure4'),
    [dash.dependencies.Input('checklist', 'value')])

# Step 3
# Define the graph with plotly express
def update_figure(id_student):
    figure1 = px.scatter(data_merged3,
                     x="id_student",
                     y="pred_fail",
                     color="id_student",
                     hover_name='id_student')
    return figure1 # You must return all the output(s) in step 2

def update_figure(id_student):
    figure2 = px.scatter(data_merged3,
                     x="id_student",
                     y="Quiz_2",
                     color="id_student",
                     hover_name='id_student')
    return figure2 # You must return all the output(s) in step 2

def update_figure(id_student):
    figure3 = px.scatter(data_merged3,
                     x="id_student",
                     y="Test_2",
                     color="id_student",
                     hover_name='id_student')
    return figure3 # You must return all the output(s) in step 2

def update_figure(id_student):
```

```
    figure4 = px.scatter(data_merged3,
                         x="id_student",
                         y="resource",
                         color="id_student",
                         hover_name='id_student')
    return figure4 # You must return all the output(s) in step 2

# Run the app
#app.run_server(mode="inline", port = 8102)
if __name__ == '__main__':
    app.run_server()
```

Dash app running on https://vcgnaoov.labs.coursera.org/proxy/8050/

In [53]:

```
# App
app = JupyterDash(__name__)
# Server
server = app.server

assess_list = data_merged3['id_student'].unique()


# Layout components
app.layout = html.Div([
    html.Div([
        html.H1('Prediction of Failure Plot'),
        dcc.Graph(id = 'graph1')]),
    html.Div([
        html.H1("Quiz 1 Plot"),
        dcc.Graph(id = 'graph2')]),
    html.Div([
        html.H1('Quiz 2 Plot'),
        dcc.Graph(id = 'graph3')]),
    html.Div([
        html.H1('Resources-Used Plot'),
        dcc.Graph(id = 'graph4')]),
    # Create a dropdown menu based on code_module
    dcc.Dropdown(
    id='dropdown_id_student',
    options=[{'label': i, 'value': i} for i in assess_list],
    value=assess_list
    )])

# Callbacks
@app.callback(
    [dash.dependencies.Output('graph1', 'figure1'),
    dash.dependencies.Output('graph2', 'figure2'),
    dash.dependencies.Output('graph3', 'figure3'),
    dash.dependencies.Output('graph4', 'figure4')],
    [dash.dependencies.Input('dropdown_id_student', 'value')])

def update_figure(id_student):
    figure1 = px.bar(data_merged3, x = 'id_student', y = 'pred_fail')
    return figure1

def update_figure(id_student):
    figure2 = px.bar(data_merged3, x = 'id_student', y = 'Quiz_2')
    return figure2

def update_figure(id_student):
    figure3 = px.bar(data_merged3, x = 'id_student', y = 'Test_2')
    return figure3

def update_figure(id_student):
    figure4 = px.bar(data_merged3, x = 'id_student', y = 'resource')
    return figure4

#suppress_callback_exceptions=True
# Run
#app.run_server(mode = 'inline', port = 8153)

if __name__ == '__main__':
    app.run_server()
```

Dash app running on https://vcgnaoov.labs.coursera.org/proxy/8050/

In [54]:

```
external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']
# App
app = JupyterDash(__name__, external_stylesheets=external_stylesheets)
# Server
server = app.server

# Create a unique list of code_module
assess_list = data_merged3['id_student'].unique()

app.layout = html.Div([
    # Create a html title for the dashboard
    html.H1("Prediction of Failure Plot"),

    # Create a graph, we will configure the graph using plotly express in step 3
    dcc.Graph(id='graph1', figure={}),
    dcc.Graph(id='graph2', figure={}),
    dcc.Graph(id='graph3', figure={}),
    dcc.Graph(id='graph4', figure={}),

    dcc.Checklist(  id='pred_fail',
                    options=[{'label': i, 'value': i} for i in assess_list],
                    value=data_merged3['pred_fail']),

    dcc.Checklist(  id='Quiz_2',
                    options=[{'label': i, 'value': i} for i in assess_list],
                    value=data_merged3['Quiz_2']),

    dcc.Checklist(  id='Test_2',
                    options=[{'label': i, 'value': i} for i in assess_list],
                    value=data_merged3['Test_2']),

    dcc.Checklist(  id='resource',
                    options=[{'label': i, 'value': i} for i in assess_list],
                    value=data_merged3['resource'])



])


# Callbacks
@app.callback(
    dash.dependencies.Output('graph1','figure'),
    dash.dependencies.Output('graph2', 'figure'),
    dash.dependencies.Output('graph3', 'figure'),
```

```
    dash.dependencies.Output('graph4', 'figure'),
    [dash.dependencies.Input('pred_fail', 'id_student')],
    [dash.dependencies.Input('Quiz_2', 'id_student')],
    [dash.dependencies.Input('Test_2', 'id_student')],
    [dash.dependencies.Input('resource', 'id_student')])


def update_figure():
    #figure1 = px.bar(data_merged3, x = 'id_student', y = 'pred_fail')
    figure = px.scatter(data_merged3,
                        x='id_student',
                        y='pred_fail',
                        color="id_student",
                        hover_name='id_student')
    return [figure]
def update_figure(id_student):
    #figure2 = px.bar(data_merged3, x = 'id_student', y = 'Quiz_2')
    figure = px.scatter(data_merged3,
                        x='id_student',
                        y='Quiz_2',
                        color="id_student",
                        hover_name='id_student')
    return [figure]
def update_figure(id_student):
    #figure3 = px.bar(data_merged3, x = 'id_student', y = 'Test_2')
    figure = px.scatter(data_merged3,
                        x='id_student',
                        y='Test_2',
                        color="id_student",
                        hover_name='id_student')
    return [figure]
def update_figure(id_student):
#    #figure4 = px.bar(data_merged3, x = 'id_student', y = 'resource')
    figure = px.scatter(data_merged3,
                        x='id_student',
                        y='resource',
                        color="id_student",
                        hover_name='id_student')
    return [figure]

    #return figure1, figure2, figure3, figure4

# Run
#app.run_server(mode = 'inline', port = 8101)

if __name__ == '__main__':
    app.run_server()
```

```
Dash app running on https://vcgnaoov.labs.coursera.org/proxy/8050/
```

In [ ]:

In [73]:
```
def update_figure():
    figure = px.scatter(data_merged3,
                        x='id_student',
                        y='pred_fail',
                        color="id_student",
                        hover_name='id_student')
    return figure
update_figure()
```

In [74]:
```
def update_figure():
    figure = px.scatter(data_merged3,
                        x='id_student',
                        y='Quiz_2',
                        color="id_student",
                        hover_name='id_student')
    return figure
update_figure()
```

In [75]:
```
def update_figure():
    figure = px.scatter(data_merged3,
                        x='id_student',
                        y='Test_2',
                        color="id_student",
                        hover_name='id_student')
    return figure
update_figure()
```

In [86]:
```
def update_figure():
    figure = px.scatter(data_merged3,
                        x='id_student',
                        y='resource',
                        color="id_student",
                        hover_name='id_student')
    return figure
update_figure()
```

In [184]:
```
#This, and everything above, is the code I'm currently trying to get to work.  It worked fine when I only had one variable,
# one dcc.Graph, one dcc.Checklist, and one output/input under the dash callback.  I tested the individual graph codes,
# visible towards the bottom of this cell, and they all run fine individually in separate cells.  Something is still off,
# or missing, with the pieces of this code that are supposed to connect the graphs together.  Current ouput with this code
# is 5 blank graphs.

# YOUR CODE HERE
# Create a unique list of assignment_name
assess_list = data_merged2['id_student']

# Step 1
app.layout = html.Div([
            # Create a html title for the dashboard
            html.H1("Prediction of Failure Plot"),

            # Create a graph, we will configure the graph using plotly express in step 3
            dcc.Graph(id='graph1'),
            dcc.Graph(id='graph2'),
            dcc.Graph(id='graph3'),
            dcc.Graph(id='graph4'),
            #dcc.Graph(id='graph5'),

            # Create a checklist based on assignment_name
            dcc.Checklist(
                    id = 'checklist1',
                    options=[{'label1': i, 'value1': i} for i in assess_list],
                    value=assess_list # Default values contain all assignment_name
            ),
```

```
          dcc.Checklist(
                  id = 'checklist2',
                  options=[{'label2': i, 'value2': i} for i in assess_list],
                  value=assess_list # Default values contain all assignment_name
           ),


           dcc.Checklist(
                  id = 'checklist3',
                  options=[{'label3': i, 'value3': i} for i in assess_list],
                  value=assess_list # Default values contain all assignment_name
           ),


           dcc.Checklist(
                  id = 'checklist4',
                  options=[{'label4': i, 'value4': i} for i in assess_list],
                  value=assess_list # Default values contain all assignment_name
           ),


           #dcc.Checklist(
           #        id = 'checklist5',
           #        options=[{'label': i, 'value': i} for i in assess_list],
           #        value='freq_of_resource_use_compared_to_avg' # Default values contain all assignment_name
           #)
])

# Step 2
# Callback using inputs from the checklist to generate the graph
@app.callback(
    dash.dependencies.Output('graph1', 'figure1'),
    dash.dependencies.Output('graph2', 'figure2'),
    dash.dependencies.Output('graph3', 'figure3'),
    dash.dependencies.Output('graph4', 'figure4'),
    #dash.dependencies.Output('graph5', 'figure5'),
    [dash.dependencies.Input('checklist1', 'value1')],
    [dash.dependencies.Input('checklist2', 'value2')],
    [dash.dependencies.Input('checklist3', 'value3')],
    [dash.dependencies.Input('checklist4', 'value4')]
    #[dash.dependencies.Input('checklist5', 'value')]
    )

# Step 3
# Define the graph with plotly express
def update_figure(checklist1, checklist2, checklist3, checklist4):
    figure1 = px.scatter(data_merged2,
                     x='id_student',
                     y='pred_fail',
                     color="id_student",
                     hover_name='id_student')
    return figure1 # You must return all the output(s) in step 2

    figure2 = px.scatter(data_merged2,
                     x='id_student',
                     y='Quiz 2',
                     color="id_student",
                     hover_name='id_student')
    return figure2 # You must return all the output(s) in step 2

    figure3 = px.scatter(data_merged2,
                     x='id_student',
                     y='Test 2',
                     color="id_student",
                     hover_name='id_student')
    return figure3 # You must return all the output(s) in step 2

    figure4 = px.scatter(data_merged2,
                     x='id_student',
                     y='resource',
                     color="id_student",
                     hover_name='id_student')
    return figure4 # You must return all the output(s) in step 2




#def update_figure2(checklist2, id_student):
#    figure2 = px.scatter(data_merged2,
#                     x='id_student',
#                     y='Quiz 2',
#                     color="id_student",
#                     hover_name='id_student')
#    return figure2 # You must return all the output(s) in step 2

#def update_figure3(checklist3, id_student):
#    figure3 = px.scatter(data_merged2,
#                     x='id_student',
#                     y='Test 2',
#                     color="id_student",
#                     hover_name='id_student')
#    return figure3 # You must return all the output(s) in step 2

#def update_figure4(checklist4, id_student):
#    figure4 = px.scatter(data_merged2,
#                     x='id_student',
#                     y='resource',
#                     color="id_student",
#                     hover_name='id_student')
#    return figure4 # You must return all the output(s) in step 2

#def update_figure(assess_list):
#    figure5 = px.scatter(data_merged2,
#                     x='id_student',
#                     y='freq_of_resource_use_compared_to_avg',
#                     color="id_student",
#                     hover_name='id_student')
#    return figure5 # You must return all the output(s) in step 2


# Run the app
app.run_server(mode="inline", port = 8196)
```

```
---------------------------------------------------------------------------
SchemaTypeValidationError                 Traceback (most recent call last)
File /opt/conda/lib/python3.8/site-packages/flask/app.py:1523, in Flask.full_dispatch_request(self=<Flask '__main__'>)
   1521     rv = self.preprocess_request()
   1522     if rv is None:
```

```
-> 1523         rv = self.dispatch_request()
        rv = None
        self = <Flask '__main__'>
   1524 except Exception as e:
   1525     rv = self.handle_user_exception(e)

File /opt/conda/lib/python3.8/site-packages/flask/app.py:1509, in Flask.dispatch_request(self=<Flask '__main__'>)
   1507     return self.make_default_options_response()
   1508 # otherwise dispatch to the handler for that endpoint
-> 1509 return self.ensure_sync(self.view_functions[rule.endpoint])(**req.view_args)
        req = <Request 'http://vcgnaoov.labs.coursera.org/_dash-update-component' [POST]>
        rule = <Rule '/_dash-update-component' (POST, OPTIONS) -> /_dash-update-component>
        self.view_functions[rule.endpoint] = <bound method Dash.dispatch of <jupyter_dash.jupyter_app.JupyterDash object a
t 0x7fd9568f96a0>>
```

In [165]:

```
# Create a unique list of assignment_name
assess_list = data_merged2['id_student'].unique()

# Step 1
app.layout = html.Div([
            # Create a html title for the dashboard
            html.H1("Prediction of Failure Plot"),

            # Create a graph, we will configure the graph using plotly express in step 3
            dcc.Graph(id='graph'),

            # Create a checklist based on assignment_name
            dcc.Checklist(
                    id = 'checklist',
                    options=[{'label': i, 'value': i} for i in assess_list],
                    value=assess_list # Default values contain all assignment_name
            ),
])

# Step 2
# Callback using inputs from the checklist to generate the graph
@app.callback(
    dash.dependencies.Output('graph', 'figure'),
    [dash.dependencies.Input('checklist', 'value')]
    )

# Step 3
# Define the graph with plotly express
def update_figure(id_student):
    figure = px.scatter(data_merged2,
                     x='id_student',
                     y='pred_fail',
                     color="id_student",
                     hover_name='id_student')
    return figure # You must return all the output(s) in step 2
# Run the app
app.run_server(mode="inline", port = 8189)
```

In [166]:

```
# Create a unique list of assignment_name
assess_list = data_merged2['id_student'].unique()

# Step 1
app.layout = html.Div([
            # Create a html title for the dashboard
            html.H1("Prediction of Failure Plot"),

            # Create a graph, we will configure the graph using plotly express in step 3
            dcc.Graph(id='graph'),

            # Create a checklist based on assignment_name
            dcc.Checklist(
                    id = 'checklist',
                    options=[{'label': i, 'value': i} for i in assess_list],
                    value=assess_list # Default values contain all assignment_name
            ),

])

# Step 2
# Callback using inputs from the checklist to generate the graph
@app.callback(
    dash.dependencies.Output('graph', 'figure'),
    [dash.dependencies.Input('checklist', 'value')]
    )

def update_figure(id_student):
    figure = px.scatter(data_merged2,
                     x='id_student',
                     y='Quiz 2',
                     color="id_student",
                     hover_name='id_student')
    return figure # You must return all the output(s) in step 2

# Run the app
app.run_server(mode="inline", port = 8190)
```

In [167]:

```
# Create a unique list of assignment_name
assess_list = data_merged2['id_student'].unique()

# Step 1
app.layout = html.Div([
            # Create a html title for the dashboard
            html.H1("Prediction of Failure Plot"),

            # Create a graph, we will configure the graph using plotly express in step 3
            dcc.Graph(id='graph'),

            # Create a checklist based on assignment_name
            dcc.Checklist(
                    id = 'checklist',
                    options=[{'label': i, 'value': i} for i in assess_list],
                    value=assess_list # Default values contain all assignment_name
            ),

])

# Step 2
# Callback using inputs from the checklist to generate the graph
@app.callback(
    dash.dependencies.Output('graph', 'figure'),
    [dash.dependencies.Input('checklist', 'value')]
    )
# Step 3
# Define the graph with plotly express
```

```
def update_figure(id_student):
    filtered_score = data_merged2[(data_merged2.id_student == id_student)]
    figure = px.bar(filtered_score, x = 'id_student', y = 'Test 2')
    return figure # You must return all the output(s) in step 2

#def update_figure(assess_list):
#    figure3 = px.scatter(data_merged2,
#                         x='id_student',
#                         y='Test 2',
#                         color="id_student",
#                         hover_name='id_student')
#    return figure3 # You must return all the output(s) in step 2

# Run the app
app.run_server(mode="inline", port = 8191)
```

In [84]:

```
# Create a unique list of assignment_name
assess_list = data_merged2['id_student'].unique()

# Step 1
app.layout = html.Div([
            # Create a html title for the dashboard
            html.H1("Prediction of Failure Plot"),

            # Create a graph, we will configure the graph using plotly express in step 3
            dcc.Graph(id='graph'),

            # Create a checklist based on assignment_name
            dcc.Checklist(
                    id = 'checklist',
                    options=[{'label': i, 'value': i} for i in assess_list],
                    value=assess_list # Default values contain all assignment_name
            ),

])

# Step 2
# Callback using inputs from the checklist to generate the graph
@app.callback(
    dash.dependencies.Output('graph', 'figure'),
    [dash.dependencies.Input('checklist', 'value')]
    )

def update_figure(id_student):
    figure4 = px.scatter(data_merged2,
                         x='id_student',
                         y='resource',
                         color="id_student",
                         hover_name='id_student')
    return figure4 # You must return all the output(s) in step 2

# Run the app
app.run_server(mode="inline", port = 8192)
```

In [ ]:

```
# Create a unique list of assignment_name
#assess_list = data_merged2['id_student']

# Step 1
#app.layout = html.Div([
#            # Create a html title for the dashboard
#            html.H1("Prediction of Failure Plot"),
#
#            # Create a graph, we will configure the graph using plotly express in step 3
#            dcc.Graph(id='graph'),
#
#            # Create a checklist based on assignment_name
#            dcc.Checklist(
#                    id = 'checklist',
#                    options=[{'label': i, 'value': i} for i in assess_list],
#                    value='pred_fail' # Default values contain all assignment_name
#            ),
#
#])

# Step 2
# Callback using inputs from the checklist to generate the graph
##@app.callback(
 #   dash.dependencies.Output('graph', 'figure'),
 #   [dash.dependencies.Input('checklist', 'value')]
 #   )
```

In [ ]:

```
#This cell, and the next 4 cells, are just me testing out the different codes I have for the graphs above to make sure they
# work while running them individually.

#def update_figure(assess_list):
#    figure = px.scatter(data_merged2,
#                         x='id_student',
#                         y='freq_of_resource_use_compared_to_avg',
#                         color="id_student",
#                         hover_name='id_student')
#    return figure # You must return all the output(s) in step 2

#update_figure(assess_list)
```

In [ ]:

```
#px.scatter(data_merged2,
#                         x='id_student',
#                         y='pred_fail',
#                         color="id_student",
#                         hover_name='id_student')
```

In [ ]:

```
#def update_figure(assess_list):
#    figure4 = px.scatter(data_merged2,
#                         x='id_student',
#                         y='resource',
#                         color="id_student",
#                         hover_name='id_student')
#    return figure4 # You must return all the output(s) in step 2

#update_figure(assess_list)
```

In [ ]:

```
#def update_figure(assess_list):
#    figure2 = px.scatter(data_merged2,
#                         x='id_student',
#                         y='Quiz 2',
#                         color="id_student",
#                         hover_name='id_student')
#    return figure2 # You must return all the output(s) in step 2
```

```
#update_figure(assess_list)
```

In [ ]:
```
#def update_figure(assess_list):
#    figure3 = px.scatter(data_merged2,
#                         x='id_student',
#                        y='Test 2',
#                         color="id_student",
#                         hover_name='id_student')
#    return figure3 # You must return all the output(s) in step 2

#update_figure(assess_list)
```

In [ ]:
```
#The code below, from here, is the previous model I attempted to use but couldn't get to produce anything.  When running
# the app.run_server(mode="inline", port = 8111) I would receive errors about the variable 'mode' not existing.  When
# running app.run_server(port = 8153) or if __name__ == '__main__':
#                                             app.run_server()       it would run infinitely and never produce anything.


#fail_preds1 = data_merged2['pred_fail']
#resource_usage = lr_wide_sum['resource']
#foructa = data_merged2['freq_of_resource_use_compared_to_avg']
#Quiz_2 = data_merged2['Quiz 2']
#Test_2 = data_merged2['Test 2']
#id_student = data_merged2['id_student']


#analysis_df = pd.DataFrame({'id_student': id_student,
#                        'fail_predictions': fail_preds1,
#                        'Quiz 2': Quiz_2,
#                        'Test 2': Test_2,
#                        'foructa': foructa,
#                        'resource_usage': resource_usage})

#analysis_df.head()
```

In [ ]:
```
#!python -m venv ~/.venv/dash
#!source ~/.venv/dash/bin/activate
#!python -m pip install --upgrade pip
#!pip install jupyter-dash
#!pip install jupyterlab
#!jupyter-Lab
#!python -m pip install --upgrade jupyterlab
```

In [ ]:
```
# Standard Imports
#import dash
#from dash import dcc
#from dash import html
#import pandas as pd
#import plotly.graph_objects as go

#fail_preds1 = data_merged2['pred_fail']
#resource_usage = lr_wide_sum['resource']
#Quiz_2 = data_merged2['Quiz 2']
#Test_2 = data_merged2['Test 2']
#id_student = data_merged2['id_student']
#foructa = data_merged2['freq_of_resource_use_compared_to_avg']

#analysis_df = pd.DataFrame({'id_student': id_student,
#                        'fail_predictions': fail_preds1,
#                        'Quiz 2': Quiz_2,
#                        'Test 2': Test_2,
#                        'Frequency of Resource Use Compared to Average': foructa})


# ======================== Setting the margins
#layout = go.Layout(
#    margin=go.layout.Margin(
#        l=40,  # left margin
#        r=40,  # right margin
#        b=10,  # bottom margin
#        t=35  # top margin
#    )
#)
```

In [ ]:
```
# ======================== Plotly Graphs
#def get_bar_chart():
#    barChart = dcc.Graph(figure=go.Figure(layout=layout).add_trace(go.Bar(x=analysis_df['id_student'],
#                                                                          y=analysis_df['Quiz 2'],
#                                                                          marker=dict(color='#351e15'))).update_layout(
#        title='Quiz 2 Analysis', plot_bgcolor='rgba(0,0,0,0)'),
#        style={'width': '50%', 'height': '40vh', 'display': 'inline-block'})
#    return barChart
```

In [ ]:
```
#get_bar_chart()
```

In [ ]:
```
# ======================== Plotly Graphs
#def get_line_chart1():
#    lineChart = dcc.Graph(figure=go.Figure(layout=layout).add_trace(go.Scatter(x=analysis_df['id_student'],
#                                                                               y=analysis_df['fail_predictions'],
#                                                                               marker=dict(
#                                                                                   color='#351e15'))).update_layout(
#        title='Course Failure Predictions', plot_bgcolor='rgba(0,0,0,0)'),
#        style={'width': '50%', 'height': '40vh', 'display': 'inline-block'})
#    return lineChart
```

In [ ]:
```
#get_line_chart1()
```

In [ ]:
```
# ======================== Plotly Graphs
#def get_scatter_plot():
#    scatterPlot = dcc.Graph(figure=go.Figure(layout=layout).add_trace(go.Scatter(x=analysis_df['id_student'],
#                                                                                 y=analysis_df['Test 2'],
#                                                                                 marker=dict(
#                                                                                     color='#351e15'),
#                                                                                 mode='markers')).update_layout(
#        title='Test 2 Analysis', plot_bgcolor='rgba(0,0,0,0)'),
#        style={'width': '50%', 'height': '40vh', 'display': 'inline-block'})
#    return scatterPlot
```

In [ ]:
```
#get_scatter_plot()
```

In [ ]:
```
# ======================== Plotly Graphs
#def get_line_chart2():
#   lineChart = dcc.Graph(figure=go.Figure(layout=layout).add_trace(go.Scatter(x=analysis_df['id_student'],
#                                                                              y=analysis_df['Frequency of Resource Use Comp
#                                                                              marker=dict(
#                                                                                  color='#351e15'))).update_layout(
```

```
#        title='Frequency of Resource Use Compared to Average', plot_bgcolor='rgba(0,0,0,0)'),
#        style={'width': '50%', 'height': '40vh', 'display': 'inline-block'})
#    return lineChart
```

In [ ]:
```
#get_line_chart2()
```

In [ ]:
```
#import plotly.express as px
```

In [ ]:
```
# ======================== Dash App
#app = JupyterDash(__name__)
# Create server variable
#server = app.server
# ======================== App Layout
#app.layout = html.Div([
#    html.H1('Student Analytics Dashboard'),  #style={'text-align': 'center', 'background-color': '#ede9e8'}
#    dcc.Graph(id='my_bar_graph'), #style={'width': '50%', 'height': '40vh', 'display': 'inline-block'}
    #get_line_chart1(),
    #get_scatter_plot(),
    #get_line_chart2()
#])

# Step 2
# Callback using inputs from the checklist to generate the graph
#@app.callback(
#    dash.dependencies.Output('my_bar_graph', 'figure')
#    )
#def get_bar_chart():
#    figure = px.scatter(analysis_df, x= 'id_student', y = 'Quiz 2')
#     figure=go.Figure().add_trace(go.Bar(x=analysis_df['id_student'],
#                                                                                y=analysis_df['Quiz 2'],
#                                                                                marker=dict(color='#351e15'))).update_layout(
#         title='Quiz 2 Analysis', plot_bgcolor='rgba(0,0,0,0)')

#    return figure # You must return all the output(s) in step 2
# Run the app
#app.run_server(mode="inline", port = 8103)
```

In [ ]:
```
#figure = px.scatter(analysis_df, x= 'id_student', y = 'Quiz 2')
#figure
```

In [ ]:
```
#if __name__ == '__main__':
#    app.run_server()
```

In [ ]:
```
#app.run_server(mode="inline", port = 8111)
#app.run_server()
```

In [ ]:
```
#app.run_server(port = 8153)
```

In [ ]:
```
# Run the app
#app.run_server(mode="jupyterlab", port = 8111)
```

2. Interpret (10 points)

After developing your integrated dashboard, provide your thoughs on the following questions:

2.1 How well will your integrated dashboard help an instructor identify which students to check-in with, and why? (5 points)

My plotly dashboard(s) provide teachers with the ability to observe which students are utilizing class-resources above-average vs below-average, the rate at which the resources are used by the students, the rate at which students performed on Quiz 2, the rate at which students performed on Test 2, and which students are predicted to potentially fail. The ability to observe and identify these factors can help a teacher determine which students to pay closer attention to, and assist the teacher in spotting potential struggles students may be having. Observing the amount resources are being utilized can help a teacher determine how much the current class of students are trying to learn, as well as observe whether or not a new approach, rather than extra resources online, is needed for the current group of students. Viewing the students' performances for Test 2 and Quiz 2 can permit the teacher to identify how well her students are acquiring the information provided during classes, and help the teacher determine what students are still struggling since Test 1. Grouping students' resource-utilization in-comparison to the class-average can help identify which students may need assistance discovering the resources available and how they can use them for extra practice.

2.2 Based on the readings and videos, what would you recommend as improvements to your dashboard, and why? (5 points)

The initial dashboard I had working only contained one variable, yet needed more than one. Was able to produce 4 graphs in the same dash at one point, but the dropdown menu did not fully function. If I had been able to develop the attempted dashboard, containing pred_fail rates, Quiz 2 scores, Test 2 scores, and resource usage...I would have also sought to discuss with teachers individually as to when they typically begin to see students at their specific grade-level (as far as which quiz/test) begin to show signs of struggling. With the goal of helping students before they are fully struggling, any improvements made to the accuracy of the data on these dashboards could assist teachers in better detecting when their students need extra help. Acquiring more data from the teachers/schools as time passes, and using larger amounts of data for these analyses and predictions, could also detect more trends/patterns in the data and create better predictions. I could have settled for the dashboard I had at one point with only the dropdown menu not functioning, but looking to improve off of what you have already developed is the only way you can help others, such as students, also improve. Helping students learn how to find the motivation and energy needed to continue to grow and learn is an even bigger issue that teachers can start addressing more than they have in the past by utilizing the data analytics resources available to help enhance their observational abilities while monitoring the students.

P.S. If it is possible to see the correct answer for setting up the plotly dash, that would be helpful for future reference. Following the examples multiple times I feel like I was very close, and was able to produce 4 graphs together with a non-functioning dropdown-menu at one point...but chose to keep trying to find the correct setup vs settling with a non-functioning drop-down menu.

Deepti replied to a thread: that would give me only one chart. You also should extrapolate this code and check why errors are occurring and fix them. For example, doing that for 2 charts will give you an integrated dashboard of 2 charts:

In [ ]:
```
# App
#2 at a time
app = JupyterDash(__name__)
# Server
server = app.server
# Layout components

unique_student_ids = data_merged3['id_student'].unique()

app.layout = html.Div([
    html.Div([
        html.H1('Prediction of Failure Plot'),
        dcc.Graph(id = 'graph1'),
        dcc.Checklist(
            id='dropdown_id_student1',
            options=[{'label': i, 'value': i} for i in unique_student_ids],
            value=unique_student_ids[:5]
        )
    ]),
```

```
    html.Div([
        html.H1("Quiz 1 Plot"),
        dcc.Graph(id = 'graph2'),
        dcc.Checklist(
            id='dropdown_id_student2',
            options=[{'label': i, 'value': i} for i in unique_student_ids],
            value=unique_student_ids
        )
    ]),
])
# Callbacks
@app.callback(
    dash.dependencies.Output('graph1', 'figure'),
    [dash.dependencies.Input('dropdown_id_student1', 'value')])
def update_figure(id_student):
    figure1 = px.scatter(data_merged3, x = 'id_student', y = 'pred_fail')
    return figure1
@app.callback(
    dash.dependencies.Output('graph2', 'figure'),
    [dash.dependencies.Input('dropdown_id_student2', 'value')])
def update_figure(id_student):
    figure2 = px.histogram(data_merged3, x = 'id_student', y = 'Quiz_2')
    return figure2

# Run
#app.run_server(mode = 'inline', port = 8153)

if __name__ == '__main__':
    app.run_server()
```