# 1. Building Interactive Dashboards for Machine Learning using Plotly Dash

Welcome to this project! We will be building an interactive dashboard in a machine learning context. This kind of dashboard can be used for exploratory data analysis, as well as for model evaluation. We will be using this dashboard for the latter: we will visualize the results of different dimensionality reduction algorithms on a customer segmentation task.

Plotly Dash gives us the capability to design a web-based dashboard that allows user input to decide what is shown on the screen. For example, we can have multiple plots that interact amongst themselves depending where the user is hovering the mouse. It can also allow for other forms of input, such as dropdowns, radio buttons, text entry, and much more.

## 1.1 Prerequisites

- Intermediate-level knowledge of Python (for example, NumPy and Pandas)
- Basic UNIX/Bash skills for launching our script
- Some understanding of HTML can be beneficial
- Experience with some plotting libraries can help some understanding, for example Matplotlib (or ideally Plotly)

## 1.2 Project Outline

**Task 1**: Introduction (this section)

**Task 2**: HTML Skeleton of Project

**Task 3**: Latent Space Scatter Plot

**Task 4**: Styling Scatter Plot

**Task 5**: Linking Bar Charts

**Task 6**: Styling Bar Charts

**Task 7**: Investigating what we've built!

## Task 1: Introduction

This is the dataset we will be using. It is collated by Margarida G. M. S. Cardoso, and comprises annual spending across different types of retail products (for example, Frozen. Grocery, Delicatessen, etc.). We will use unsupervised methods to reduce the dimensionality of this data, and plot the resulting 2-D data, and investigate what our models are learning.

### 1) Models

The models that will be in our data.

- Principal Component Analysis (PCA)
- Uniform Manifold Approximation and Projection (UMAP)
- Autoencoder (AE)
- Variational Autoencoder (VAE)

These models are not the focus of this project, but we will discuss their results by the final task.

In [1]:
```
import pandas as pd

df = pd.read_csv('customer_dataset.csv')
df.head()
```

Out[1]:

| | Channel | Region | Fresh | Milk | Grocery | Frozen | Detergents_Paper | Delicatessen | pca_x | pca_y | umap_x | umap_y | ae_x | ae_y | vae_x |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 2 | 3 | 12669 | 9656 | 7561 | 214 | 2674 | 1338 | 0.193291 | -0.305100 | 7.084310 | 6.933166 | 3.548878 | 3.811006 | 0.828640 |
| **1** | 2 | 3 | 7057 | 9810 | 9568 | 1762 | 3293 | 1776 | 0.434420 | -0.328413 | 6.252880 | 7.050780 | 3.579156 | 2.955884 | 0.838629 |
| **2** | 2 | 3 | 6353 | 8808 | 7684 | 2405 | 3516 | 7844 | 0.811143 | 0.815096 | 8.588828 | 6.877347 | 1.341199 | 2.187068 | 0.841106 |
| **3** | 1 | 3 | 13265 | 1196 | 4221 | 6404 | 507 | 1788 | -0.778648 | 0.652754 | 13.654358 | 7.857928 | 6.349530 | 8.099434 | 0.814431 |
| **4** | 2 | 3 | 22615 | 5410 | 7198 | 3915 | 1777 | 5185 | 0.166287 | 1.271434 | 9.122227 | 5.977852 | 1.150562 | 3.304798 | 0.853156 |

In [1]:
```
import dash
import dash_core_components as dcc
import dash_bootstrap_components as dbc
import dash_html_components as html
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
import numpy as np
from jupyter_dash import JupyterDash

external_stylesheets = [dbc.themes.DARKLY]
app = JupyterDash(__name__, title = 'Interactive Model Dashboard', external_stylesheets = [external_stylesheets])

df = pd.read_csv('customer_dataset.csv')
features = ['Fresh', 'Milk', 'Grocery', 'Frozen', 'Detergents_Paper', 'Delicatessen']
modles = ['PCA', 'UMAP', 'AE', 'VAE']
df_average = df[features].mean()
max_val = df[features].max().max()

app.layout = html.Div([

    html.Div([

        html.Div([

            html.Div([html.Label('Model selection')], style = {'font-size' : '18px'}),

            dcc.Dropdown(

                id = 'crossfilter-model',
                options = [
                    {'label' : 'Principal Component Analysis', 'value' : 'PCA'},
                    {'label' : 'Uniform Manifold Approximation and Projection', 'value' : 'UMAP'},
                    {'label' : 'Autoencoder', 'value' : 'AE'},
                    {'label' : 'Variational Autoencoder', 'value' : 'VAE'}
                ],
                value = 'PCA',
                clearable = False
            )
        ], style = {'width' : '49%', 'display' : 'inline-block'}),


        html.Div([

            html.Div([html.Label('Feature selection')], style = {'font-size' : '18px', 'width' : '40%', 'display' :
'inline-block'}),

            html.Div([

                dcc.RadioItems(
                    id = 'gradient-scheme',
                    options = [
                        {'label' : 'Orange to Red', 'value' : 'OrRd'},
                        {'label' : 'Viridis', 'value' : 'Viridis'},
                        {'label' : 'Plasma', 'value' : 'Plasma'},
                    ],
                    value = 'Plasma',
                    labelStyle = {'float' : 'right', 'display' : 'inline-block', 'margins-right' : 10}
                )
            ], style = {'width' : '49%', 'display' : 'inline-block', 'float' : 'right'}),

            dcc.Dropdown(
                id = 'crossfilter-feature',
                options = [{'label' : i, 'value' :i} for i in features + ['None', 'Region', 'Channel', 'Total_Spend'
]],

                value = 'None',
                clearable = False)
        ], style = {'width' : '49%', 'float' : 'right', 'display' : 'inline-block'}

        )
     ], style = {'backgroundCOlor' : 'rgb(17, 17, 17)', 'padding' : '10px 5px'}),

    html.Div([

        dcc.Graph(
            id = 'scatter-plot',
            hoverData = {'points' : [{'customdata' : 0}]}
        )
    ], style = {'width' : '100%', 'height' : '90%', 'display' : 'inline-block', 'padding' : '0 20'}),

    html.Div([

        dcc.Graph(
            id = 'point-plot'
        )
    ], style = {'display' : 'inline-block', 'width' : '100%'}),

], style = {'backgroundColor' : 'rgb(17, 17, 17)'})

@app.callback(
    dash.dependencies.Output('scatter-plot', 'figure'),
    [
        dash.dependencies.Input('crossfilter-feature', 'value'),
        dash.dependencies.Input('crossfilter-model', 'value'),
        dash.dependencies.Input('gradient-scheme', 'value')
    ]
)

def update_graph(feature, model, gradient) :

    if feature == 'None' :
        cols = None
        sizes = None
        hover_names = [f'Customer {ix}' for ix in df.index]

    elif features in ['Region', 'Channel'] :
        cols = df[feature].astpye(str)
        sizes = None
        hover_names = [f'Customer {ix}' for ix in df.index]

    else :
        cols = df[feature]
        sizes = [np.max([max_val, val]) for val in df[feature].values]
        hover_names = []
        for ix, val in zip(df.index.values, df[feature].values) :
            hover_names.append(f'Customer {ix}<br>{feature} value of {val}')

    fig = px.scatter(df,
                     x = df[f'{model.lower()}_x'],
                     y = df[f'{model.lower()}_y'],
                     opacity = 0.8,
                     template = 'plotly_dark',
                     color_continuous_scale = gradient,
                     hover_name = hover_names, color = cols, size = sizes)

    fig.update_traces(customdata = df.index)

    fig.update_layout(
        height = 650,
        hovermode = 'closest',
        template = 'plotly_dark'
    )

    fig.update_xaxes(showticklabels = False)
    fig.update_yaxes(showticklabels = False)

    return fig

def create_point_plot(df, title) :
    fig = go.Figure(
            data = [
                go.Bar(name = 'Average', x = features, y = df_average.values, marker_color = '#c178f6'),
                go.Bar(name = title, x = features, y = df.values, marker_color = '#89efbd')
            ]
    )
    fig.update_layout(
        barmode = 'group',
        height = 220,
        margin = {'l' : 20, 'b' : 30, 'r' : 10, 't' : 10},
        template = 'plotly_dark'
    )
    fig.update_xaxes(showgrid = False)
    fig.update_yaxes(type = 'log', range = [0, 5])
    return fig

@app.callback(
    dash.dependencies.Output('point-plot', 'figure'),
    [
        dash.dependencies.Input('scatter-plot', 'hoverData')
    ]
)

def update_point_plot(hoverData) :
    index = hoverData['points'][0]['customdata']
    title = f'Customer {index}'
    return create_point_plot(df[features].iloc[index], title)

app.run_server(mode='external')
```

```
Dash app running on http://127.0.0.1:8050/
```