In [1]:
```
import numpy
import scipy
import pandas
import matplotlib.pyplot as plt
import sklearn
```

Importing the customer data:

In [10]:
```
df = pandas.read_csv('Mall_Customers.csv')
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              200 non-null    int64
 1   Gender                  200 non-null    object
 2   Age                     200 non-null    int64
 3   Annual Income (k$)      200 non-null    int64
 4   Spending Score (1-100)  200 non-null    int64
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
None
```

Let's check the clustering algorithms available in sklearn:

In [11]:
```
import sklearn.cluster

print(f'List of algorithms available:\n{", ".join([a for a in dir(sklearn.cluster) if ty
pe(getattr(sklearn.cluster, a)) == type])}')
```

```
List of algorithms available:
AffinityPropagation, AgglomerativeClustering, Birch, DBSCAN, FeatureAgglomeration, KMean
s, MeanShift, MiniBatchKMeans, OPTICS, SpectralClustering
```

First things first, we need to make sure that attributes are normalized to ensure that distance metrics will work properly.

In [76]:
```
print('Before normalization')
print(df.describe())
df_numerical = df.drop(columns=['Gender'])  # don't forget about the non-numerical colum
ns!
df_n = (df_numerical-df_numerical.mean())/df_numerical.std()
df_n['Gender'] = df['Gender']
print('After normalization')
print(df_n.describe())
```

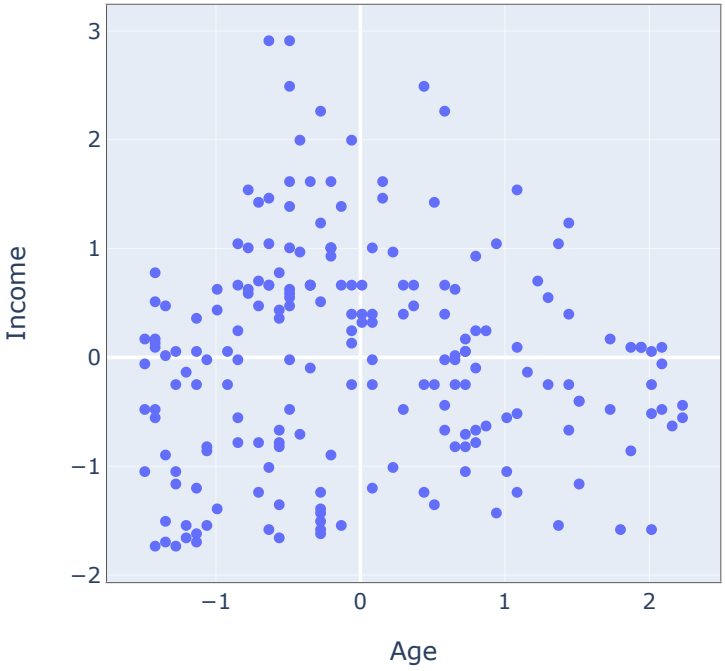
```
Before normalization
       CustomerID         Age  Annual Income (k$)  Spending Score (1-100)
count  200.000000  200.000000          200.000000              200.000000
mean   100.500000   38.850000           60.560000               50.200000
std     57.879185   13.969007           26.264721               25.823522
min      1.000000   18.000000           15.000000                1.000000
25%     50.750000   28.750000           41.500000               34.750000
50%    100.500000   36.000000           61.500000               50.000000
75%    150.250000   49.000000           78.000000               73.000000
max    200.000000   70.000000          137.000000               99.000000
After normalization
         CustomerID           Age  Annual Income (k$)  Spending Score (1-100)
count  2.000000e+02  2.000000e+02        2.000000e+02            2.000000e+02
mean   7.105427e-17 -1.021405e-16       -2.131628e-16           -1.376677e-16
std    1.000000e+00  1.000000e+00        1.000000e+00            1.000000e+00
min   -1.719098e+00 -1.492590e+00       -1.734646e+00           -1.905240e+00
25%   -8.595491e-01 -7.230292e-01       -7.256883e-01           -5.982918e-01
50%    0.000000e+00 -2.040231e-01        3.578945e-02           -7.744877e-03
75%    8.595491e-01  7.266085e-01        6.640086e-01            8.829160e-01
max    1.719098e+00  2.229937e+00        2.910368e+00            1.889750e+00
```

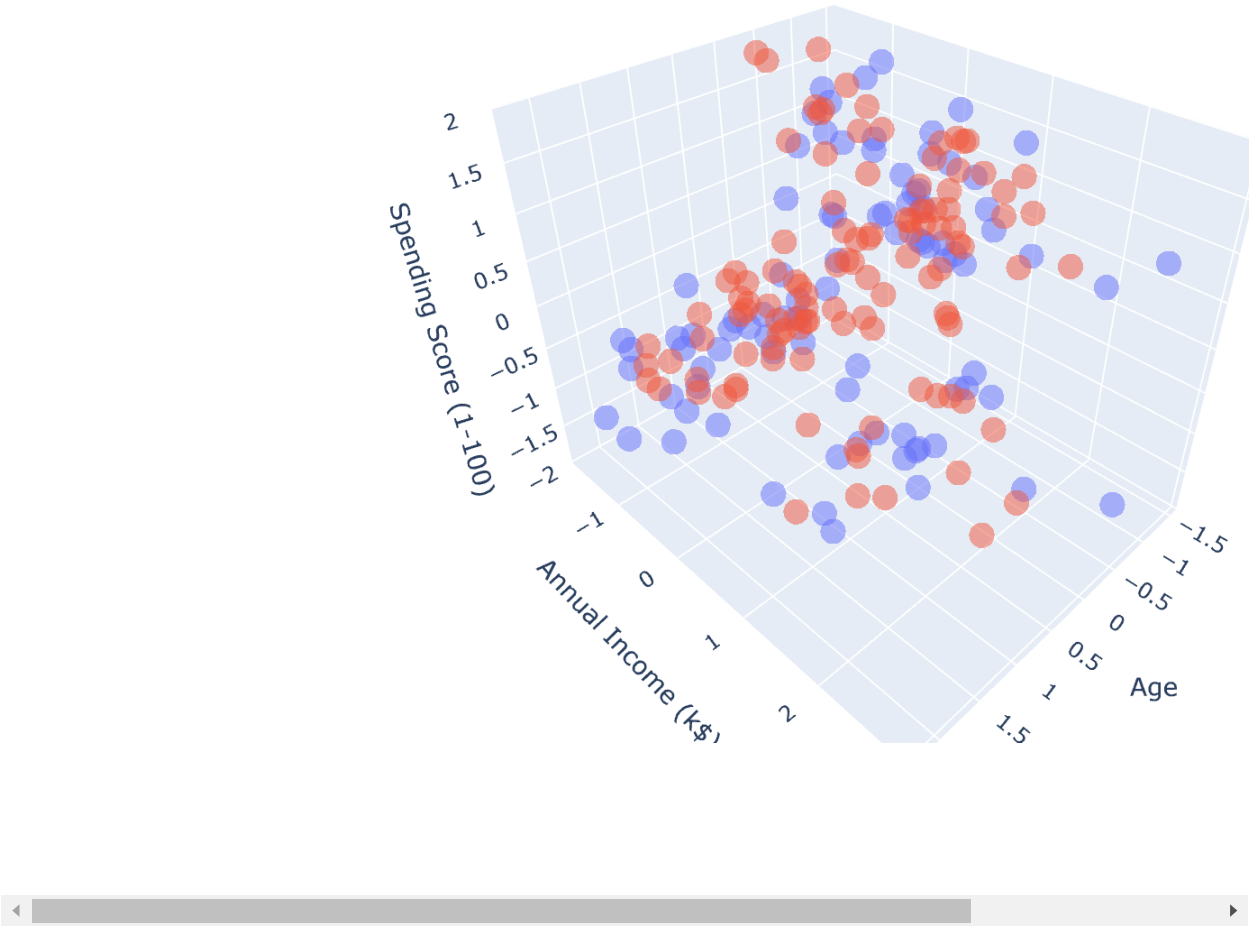
We are now ready to apply the algorithms, let's visualize the data first, using plotly:

In [152]:

```
import plotly.offline
import plotly.express
import plotly.graph_objs

plotly.offline.init_notebook_mode()
# plots
age_vs_income_scatter = plotly.graph_objs.Scatter(
    x=df_n['Age'], y=df_n['Annual Income (k$)'], mode='markers',
    )

# layout
layout = plotly.graph_objs.Layout(
    autosize=False,
    width=500,
    height=500,
    title='Age vs Income (Normalized)',
    xaxis= plotly.graph_objs.layout.XAxis(
        linecolor = 'black', linewidth = 1, mirror = True,
        title='Age'
    ),
    yaxis= plotly.graph_objs.layout.YAxis(
        linecolor = 'black', linewidth = 1, mirror = True,
        title='Income'
    )
)
fig1 = plotly.graph_objs.Figure(data=[age_vs_income_scatter], layout=layout)
fig1.show()
""" 3d plot """
fig2 = plotly.express.scatter_3d(df_n, x='Age', y='Annual Income (k$)', z='Spending Scor
e (1-100)',
                                 opacity=0.5, color='Gender')
fig2.update_layout({'height': 600})
fig2.show()
```

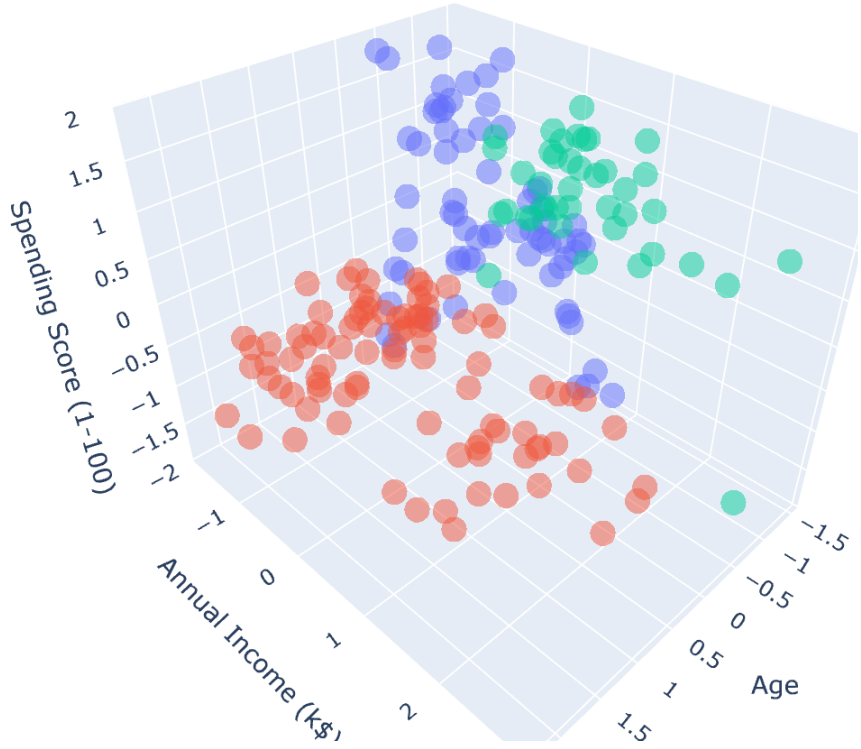
Age vs Income (Normalized)





Let's apply k-means algorithm to separate customers into three clusters:

In [139]:
```
from sklearn.cluster import KMeans

# need to handle the gender column before passing it to the k-means
df_n['IsFemale'] = df['Gender'].apply(lambda x: 1.0 if x == 'Female' else 0.0)
X = df_n.drop(columns=['CustomerID', 'Gender'])
kmeans_result = KMeans(n_clusters=3, random_state=3022022).fit(X)
# visualizing again
df_kmeans = X.copy()
df_kmeans['Cluster ID'] = kmeans_result.labels_.astype(str)
fig3 = plotly.express.scatter_3d(df_kmeans, x='Age', y='Annual Income (k$)', z='Spending
Score (1-100)',
                                 opacity=0.5, color='Cluster ID')
fig3.update_layout({'height': 600})
fig3.show()
```

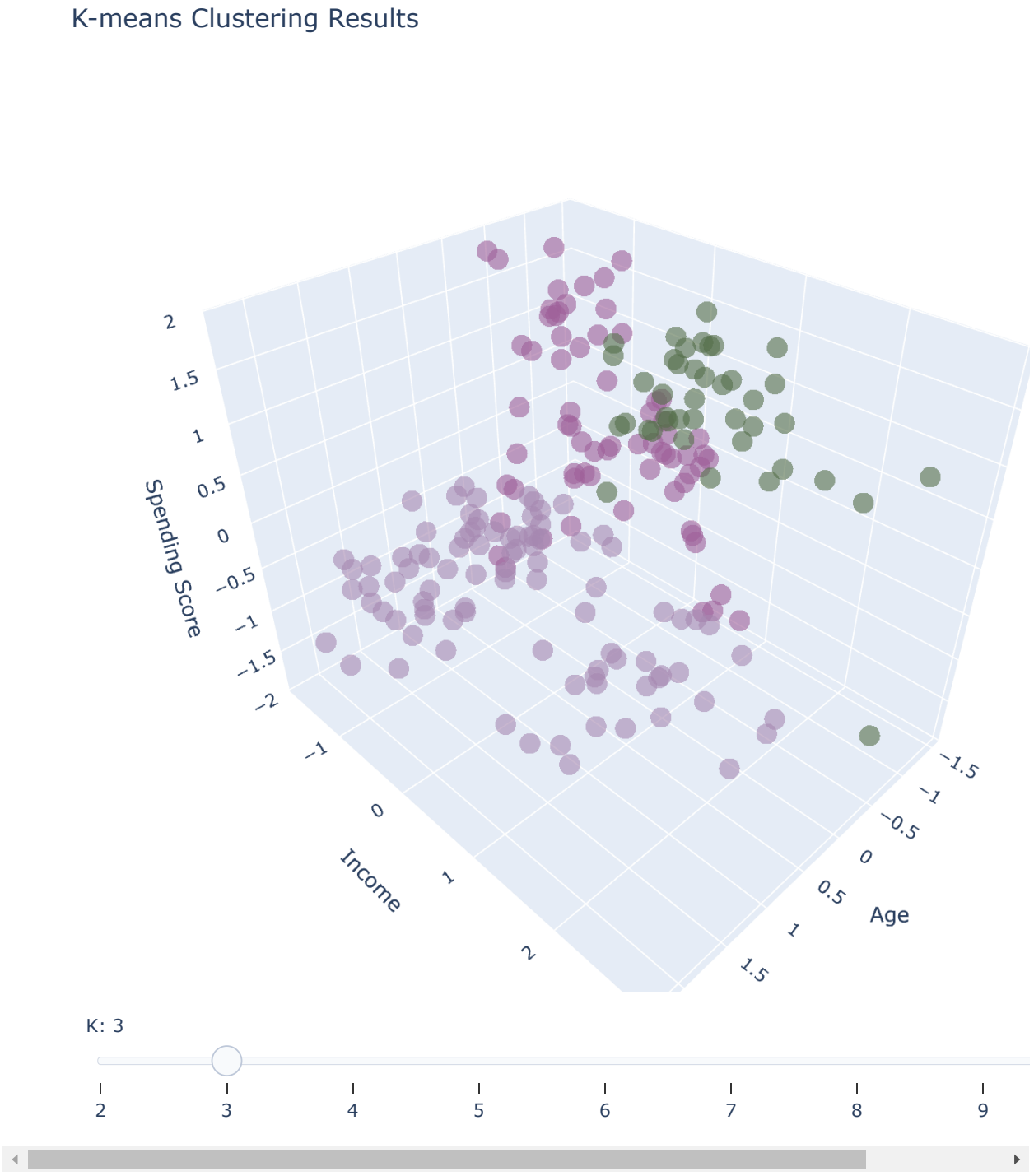


Changing the number of clusters:

In [151]:

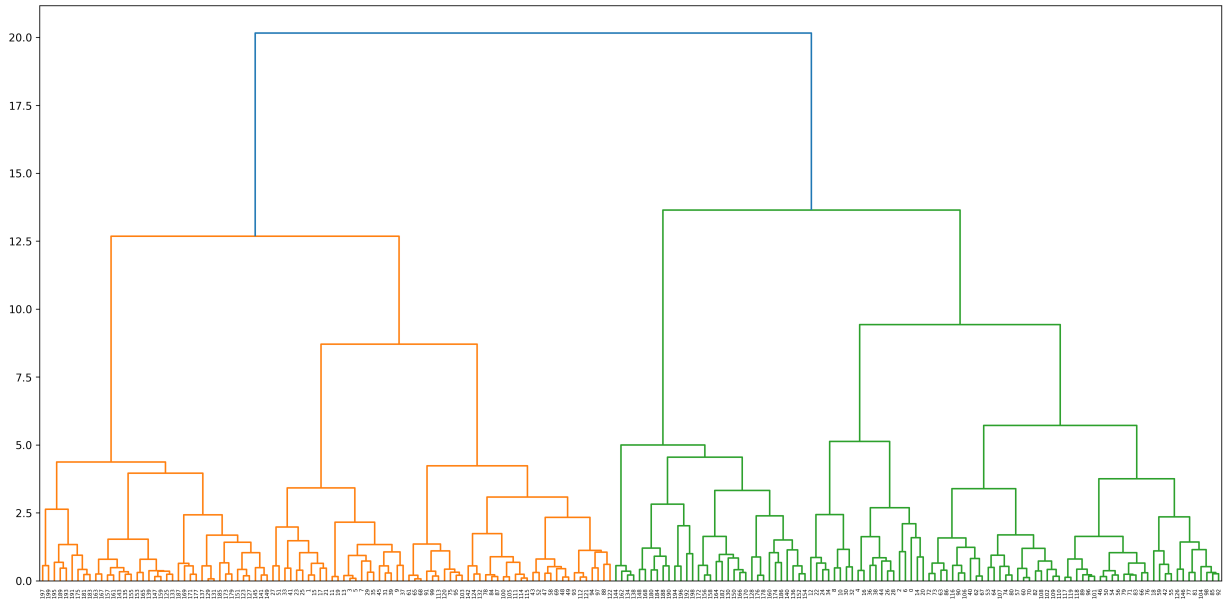
```
# code example: https://plotly.com/python/ipython-notebook-tutorial/
from plotly.graph_objs import Scatter3d

# initialize
numpy.random.seed(433)
max_nb_clusters = 10
list_of_colors = [f'rgb({r},{g},{b})'
                  for r, g, b in numpy.random.randint(0, 256, (max_nb_clusters, 3))]
min_nb_cluster = 2

# create the plots
data = [
    Scatter3d(
        visible=False,
        line=dict(color='#00CED1', width=6),
        name=f'K = {nb_clusters}',
        x=df_kmeans['Age'],
        y=df_kmeans['Annual Income (k$)'],
        z=df_kmeans['Spending Score (1-100)'],
        mode='markers',
        marker=dict(
            color=[
                list_of_colors[i]
                for i in KMeans(
                    n_clusters=nb_clusters, random_state=722
                ).fit(X).labels_
            ],
            colorscale='Viridis',
            opacity=0.6,
        ),
    ) for nb_clusters in range(min_nb_cluster, max_nb_clusters + 1)]
data[1]['visible'] = True
# configure steps
steps = []
for i in range(len(data)):
    step = dict(
        method='restyle',
        args=['visible', [False] * len(data)],
        label=str(i + min_nb_cluster)
    )
    step['args'][1][i] = True
    steps.append(step)

# add the slider
sliders = [dict(
    active=1,
    currentvalue={"prefix": "K: "},
    pad={"t": 12},
    steps=steps
)]

# layout = dict(sliders=sliders)

layout = plotly.graph_objs.Layout(
    autosize=False,
    width=800,
    height=800,
    title='Clusters overview',
    xaxis= plotly.graph_objs.layout.XAxis(
        linecolor = 'black', linewidth = 1, mirror = True,
        title='Age'
    ),
    yaxis= plotly.graph_objs.layout.YAxis(
```

```
        linecolor = 'black', linewidth = 1, mirror = True,
        title='Income'
    ),
    margin=plotly.graph_objs.layout.Margin(
        l=50,
        r=50,
        b=100,
        t=100,
        pad=4
    ),
    sliders=sliders,
)
fig4 = plotly.graph_objs.Figure(data=data, layout=layout)
fig4.update_layout(
    scene=dict(
        xaxis_title='Age',
        yaxis_title='Income',
        zaxis_title='Spending Score'),
    title_text='K-means Clustering Results',
)

plotly.offline.iplot(fig4, filename='Kmeans slider')
```



Let's see how agglomerative clustering works, we can check out the dendrogram as follows:

In [136]:
```
import scipy.cluster.hierarchy as sch
from matplotlib.pyplot import figure

figure(figsize=(20, 10), dpi=300)
dendrogram = sch.dendrogram(sch.linkage(X, method='ward'))
```



We can use the previous code to create a slider for the agglomerative clustering.

In [150]:

```
from sklearn.cluster import AgglomerativeClustering

# initialize
numpy.random.seed(433)
max_nb_clusters = 10
list_of_colors = [f'rgb({r},{g},{b})'
                  for r, g, b in numpy.random.randint(0, 256, (max_nb_clusters, 3))]
min_nb_cluster = 2

# create the plots
data = [
    Scatter3d(
        visible=False,
        line=dict(color='#00CED1', width=6),
        name=f'{nb_clusters} Clusters',
        x=df_kmeans['Age'],
        y=df_kmeans['Annual Income (k$)'],
        z=df_kmeans['Spending Score (1-100)'],
        mode='markers',
        marker=dict(
            color=[
                list_of_colors[i]
                for i in AgglomerativeClustering(
                    n_clusters=nb_clusters, affinity='euclidean', linkage='ward'
                ).fit(X).labels_
            ],
            colorscale='Viridis',
            opacity=0.6,
        ),
    ) for nb_clusters in range(min_nb_cluster, max_nb_clusters + 1)]
data[1]['visible'] = True
# configure steps
steps = []
for i in range(len(data)):
    step = dict(
        method='restyle',
        args=['visible', [False] * len(data)],
        label=str(i + min_nb_cluster)
    )
    step['args'][1][i] = True
    steps.append(step)

# add the slider
sliders = [dict(
    active=1,
    currentvalue={"suffix": " Clusters"},
    pad={"t": 12},
    steps=steps
)]

layout = plotly.graph_objs.Layout(
    autosize=False,
    width=800,
    height=800,
    xaxis= plotly.graph_objs.layout.XAxis(
        linecolor = 'black', linewidth = 1, mirror = True,
        title='Age'
    ),
    yaxis= plotly.graph_objs.layout.YAxis(
        linecolor = 'black', linewidth = 1, mirror = True,
        title='Income'
    ),
    margin=plotly.graph_objs.layout.Margin(
```

```
        l=50,
        r=50,
        b=100,
        t=100,
        pad=4
    ),
    sliders=sliders,
)
fig4 = plotly.graph_objs.Figure(data=data, layout=layout)
fig4.update_layout(
    scene=dict(
        xaxis_title='Age',
        yaxis_title='Income',
        zaxis_title='Spending Score'),
    title_text='Agglomerative Clustering Results',
)

plotly.offline.iplot(fig4, filename='Agglomerative slider')
```

Agglomerative Clustering Results