Lab 11: Unsupervised Learning with k-means

In this lab, we begin our survey of common unsupervised learning methods.

Supervised vs. Unsupervised Learning

As we know, in the supervised setting, we are presented with a set of training pairs $(\mathbf{x}^{(i)}, y^{(i)}), \mathbf{x}^{(i)} \in \mathcal{X}, y^{(i)} \in \mathcal{Y}, i \in 1..m$, where typically $\mathcal{X} = \mathbb{R}^n$ and either $\mathcal{Y} = \mathbb{R}$ (regression) or $\mathcal{Y} = \{1, \ldots, k\}$ (classification). The goal is, given a new $\mathbf{x} \in \mathcal{X}$ to come up with the best possible prediction $\hat{y} \in \mathcal{Y}$ corresponding to $\mathbf{x}$ or a set of predicted probabilities $p(y = y_i \mid \mathbf{x}), i \in \{1, \ldots, k\}$.

In the *unsupervised setting*, we are presented with a set of training items $\mathbf{x}^{(i)} \in \mathcal{X}$ without any labels or targets. The goal is generally to understand, given a new $\mathbf{x} \in \mathcal{X}$, the relationship of $\mathbf{x}$ with the training examples $\mathbf{x}^{(i)}$.

The phrase *understand the relationship* can mean many different things depending on the problem setting. Among the most common specific goals is *clustering*, in which we map the training data to K *clusters*, then, given $\mathbf{x}$, find the most similar cluster $c \in \{1, \ldots, K\}$.

k-means Clustering

Clustering is the most common unsupervised learning problem, and k-means is the most frequently used clustering algorithm. k-means is suitable when $\mathcal{X} = \mathbb{R}^n$ and Euclidean distance is a reasonable model of dissimilarity between items in $\mathcal{X}$.

The algorithm is very simple:

1. Randomly initialize k cluster centroids $\mu_1, \ldots, \mu_k \in \mathbb{R}^n$.
2. Repeat until convergence:
   A. For $i \in 1..m, c^{(i)} \leftarrow \text{argmin}_j \|\mathbf{x}^{(i)} - \mu_j\|^2$.
   B. For $j \in 1..k$,
   $$\mu_j \leftarrow \frac{\sum_{i=1}^m \delta(c^{(i)} = j)\mathbf{x}^{(i)}}{\sum_{i=1}^m \delta(c^{(i)} = j)}$$

In-Lab Exercise

Write Python code to generate 100 examples from each of three different well-separated 2D Gaussian distributions. Plot the data, initialize three arbitrary means, and animate the process of iterative cluster assignment and cluster mean assignment.

Hint:

Exercise 1.1 (5 points)

Generate 100 examples from each of **three different well-separated 2D Gaussian distributions**.
Hint:

In [17]:
```
X=None
y=None

### BEGIN SOLUTION
from sklearn.datasets import make_blobs

X, y = make_blobs(n_samples=300, centers=3, n_features=2)
### END SOLUTION
```

In [18]:
```
import numpy as np
print('X.shape', X.shape)
print('y.shape', y.shape)
print('X=\n', X[:5])
print('y=\n', y[:5])

print(y.min(), y.max())
print(len(np.unique(y)))

# Test function: Do not remove
assert X.shape == (300, 2), 'Size of X is incorrect'
assert y.shape == (300,) or y.shape == 300 or y.shape == (300,1), 
'Size of y is incorrect'
assert len(np.unique(y)) == 3, 'Number groups of samples are inco
rrect'
for i in np.unique(y):
    assert isinstance(i, np.int64) or isinstance(i, int), 'group 
type is incorrect'

print("success!")
# End Test function
```

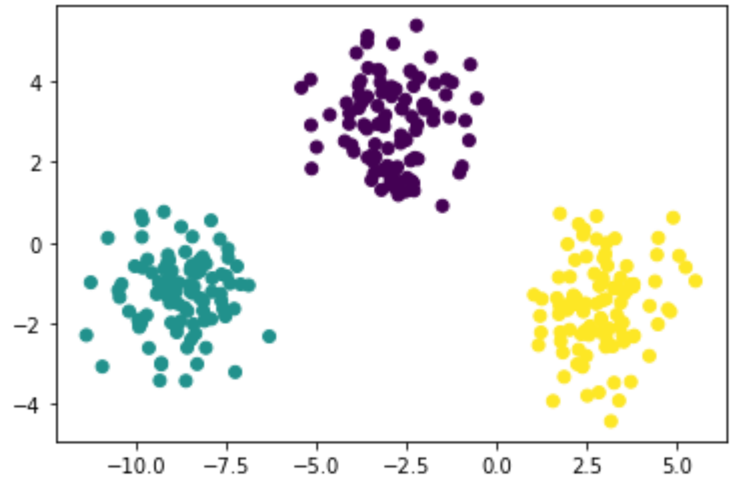
```
X.shape (300, 2)
y.shape (300,)
X=
 [[ 1.68307019 -1.37421439]
 [-2.77125555  1.31468351]
 [-2.71111912  1.44140402]
 [ 2.04684708 -0.8462684 ]
 [-9.94960684 -1.93297347]]
y=
 [2 0 0 2 1]
0 2
3
success!
```

Exercise 1.2 (5 points)

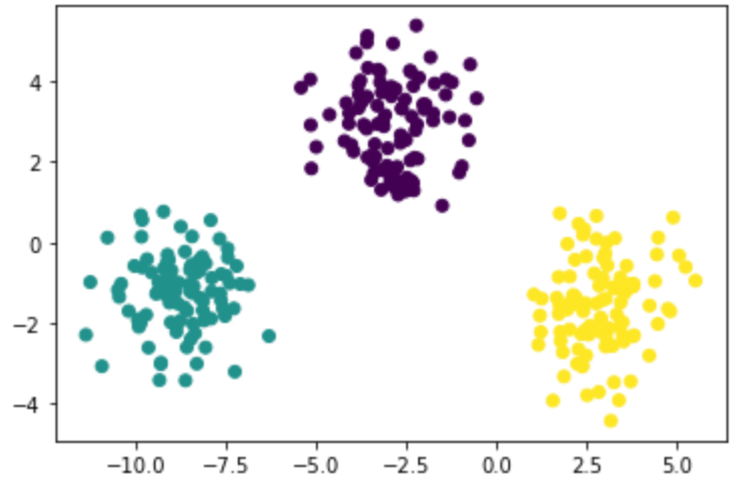
Plot the data. Separate the data by color.

In [19]:
```
import matplotlib.pyplot as plt

### BEGIN SOLUTION
plt.scatter(X[:,0],X[:,1],c=y)
### END SOLUTION
```

Out[19]: <matplotlib.collections.PathCollection at 0x7fec4c4302b0>



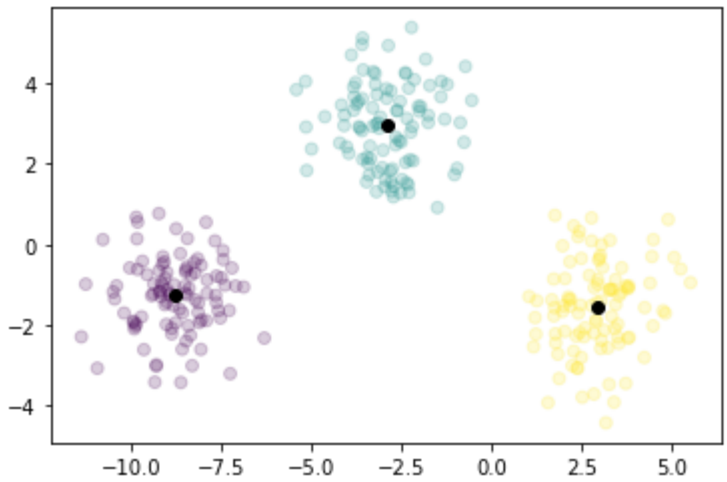
Expect result:



Exercise 1.3 (20 points)

Initialize three arbitrary means, and animate the process of iterative cluster assignment and cluster mean assignment.

In [25]:
```
import numpy as np
from IPython.display import clear_output
import time

# 1. initialize 3 random centers
centers = None
error = 9999999999.0
while True:
    # 2. find the nearest centers for each of the points

    # 3. plot the graph. Do not forget to use clear_output

    # 4. find the mean of each centers

    # 5. calculate sum square error to check error. If the error
is less than 1e-6, you can stop the loop.

    if error < 1e-6:
        break

    time.sleep(0.3)

### BEGIN SOLUTION
#initialize 3 random centers
centers = np.random.uniform(-3,3,size=(3,2))

count = 1
while True:
    #find the nearest centers for each of the points
    distance = np.empty((X.shape[0],centers.shape[0]))
    for i,x in enumerate(X):
        for j,c in enumerate(centers):
            distance[i,j] = (c-x).T@(c-x)
    nearest = np.argmin(distance,axis=1)

    clear_output(wait=True)
    plt.scatter(X[:,0],X[:,1],c=nearest,alpha=0.2)
    plt.scatter(centers[:,0],centers[:,1],c='k')
    plt.savefig(str(count) + ".png")
    count+=1
    plt.show()

    #find the mean of each centers
    mean = centers.copy()
    for i in np.unique(nearest):
        mean[i] = np.mean(X[nearest==i],axis=0)
    sqr_error = np.sum((mean-centers)**2)
    print("Error", np.sum((mean-centers)**2))
    if(np.sum((mean-centers)**2)<1e-6):
        break
    else:
        centers = mean

    time.sleep(0.1)

### END SOLUTION
```

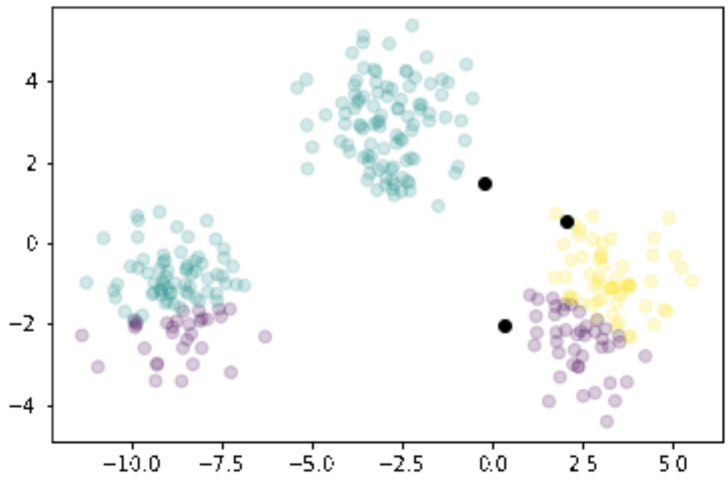



```
Error 0.0
```

In [27]:
```python
print(centers)

# Test function: Do not remove
assert centers.shape == (2, 3) or centers.shape == (3, 2), 'Size of centers is incorrect'

print("success!")
# End Test function
```

```
[[-8.80480091 -1.26618506]
 [-2.89275811  2.94505971]
 [ 2.93128093 -1.55644619]]
success!
```

Expect result:



Example with Kaggle Customer Segmentation Data

This example is based on the Kaggle Mall Customers Dataset (https://www.kaggle.com/vjchoudhary7/customer-segmentation-tutorial-in-python) and Caner Dabakoglu's (https://www.kaggle.com/cdabakoglu) tutorial on the dataset. The goal is customer segmentation.

The dataset has 5 columns, `CustomerID` , `Gender` , `Age` , `Annual Income` , and `Spending score` . We will use three of these variables, namely `Age` , `Annual Income` , and `Spending score` for segmenting customers. (Give some thought to why we don't use `CustomerID` or `Gender` .)

First, let's import some libraries:

In [28]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

Next we read the data set and print out some information about it.

In [29]:
```
df = pd.read_csv("Mall_Customers.csv")

print('Dataset information:\n')
df.info()
print('\nDataset head (first five rows):\n')
df.head()
```

```
Dataset information:

<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

Dataset head (first five rows):
```

Out[29]:

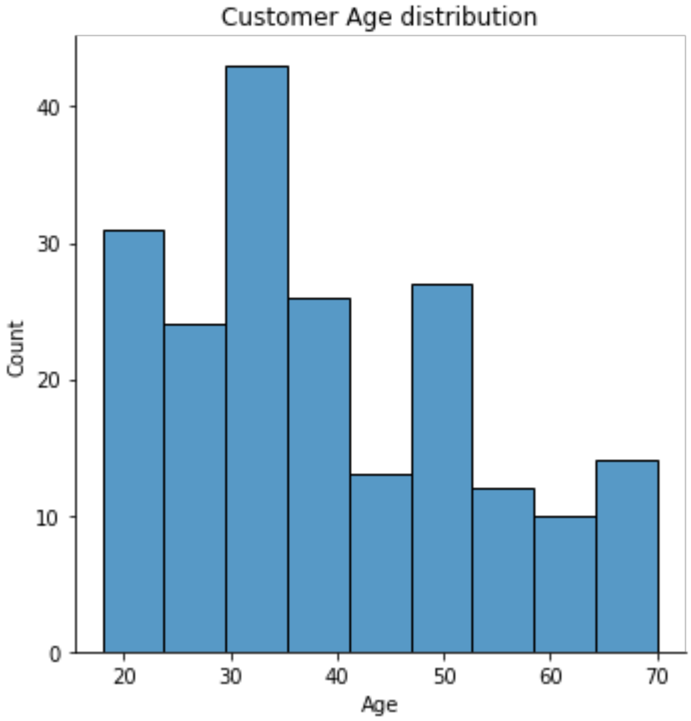
	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

Let's drop the `CustomerID` column, as it's not useful.

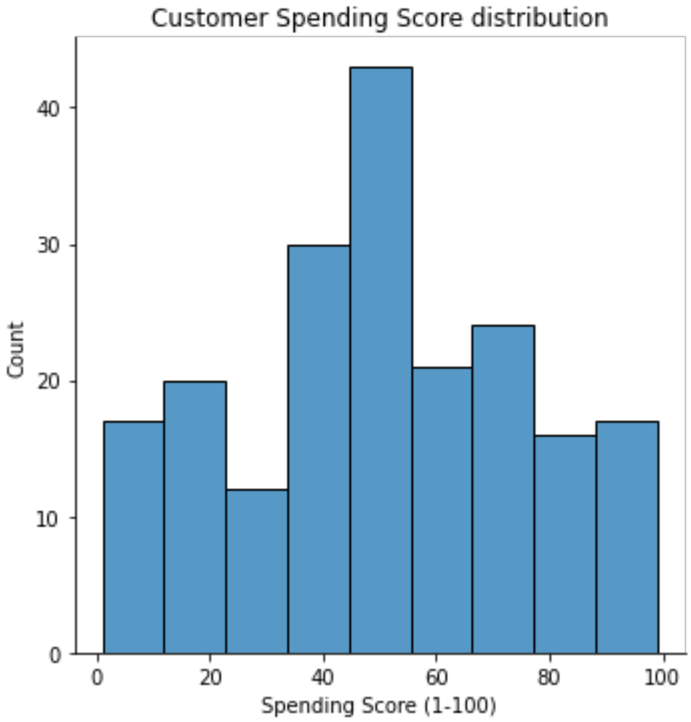
In [30]:
```
df.drop(["CustomerID"], axis = 1, inplace=True)
```

Next, let's visualize the marginal distribution over each variable, to get an idea of how cohesive they are. We can see that the variables are not quite Gaussian and have some skew:

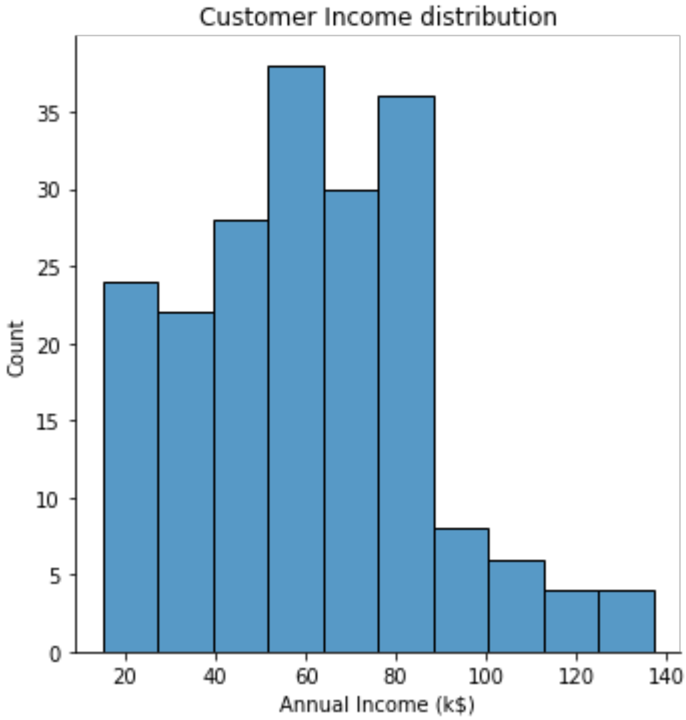
In [31]:
```
sns.displot(df.Age)
_ = plt.title('Customer Age distribution')
```



In [32]:
```
sns.displot(df['Spending Score (1-100)'])
_ = plt.title('Customer Spending Score distribution')
```

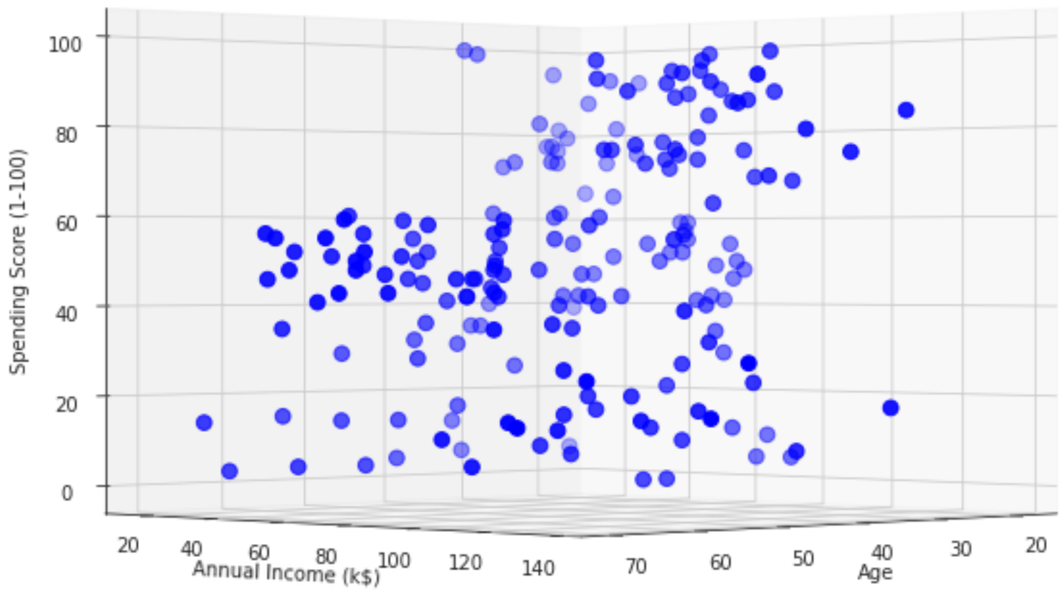


In [33]:
```
sns.displot(df['Annual Income (k$)'])
_ = plt.title('Customer Income distribution')
```



Next, let's make a 3D scatter plot of the relevant variables:

In [34]:
```
sns.set_style("white")
fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(df.Age, df["Annual Income (k$)"], df["Spending Score (1-100)"], c='blue', s=60)
ax.view_init(0, 45)
plt.xlabel("Age")
plt.ylabel("Annual Income (k$)")
ax.set_zlabel('Spending Score (1-100)')
plt.show()
```

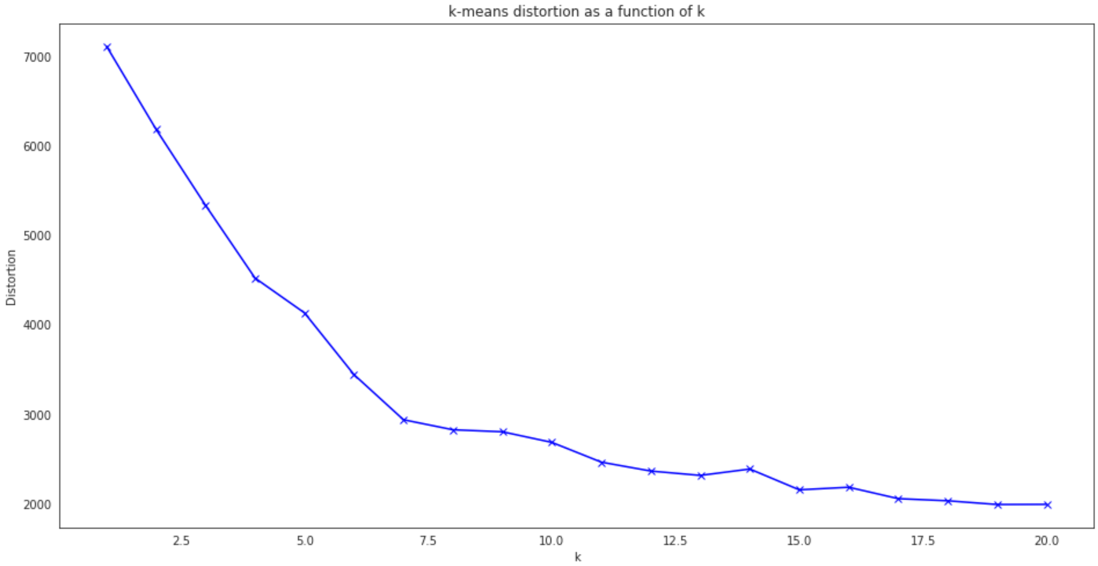


Next, let's implement k-means:

In [35]:
```
# Initialize a k-means model given a dataset

def init_kmeans(X, k):
    m = X.shape[0]
    n = X.shape[1]
    means = np.zeros((k,n))
    order = np.random.permutation(m)[:k]
    for i in range(k):
        means[i,:] = X[order[i],:]
    return means

# Run one iteration of k-means

def iterate_kmeans(X, means):
    m = X.shape[0]
    n = X.shape[1]
    k = means.shape[0]
    distortion = np.zeros(m)
    c = np.zeros(m)
    for i in range(m):
        min_j = 0
        min_dist = 0
        for j in range(k):
            dist_j = np.linalg.norm(X[i,:] - means[j,:])
            if dist_j < min_dist or j == 0:
                min_dist = dist_j
                min_j = j
        distortion[i] = min_dist
        c[i] = min_j
    for j in range(k):
        means[j,:] = np.zeros((1,n))
        nj = 0
        for i in range(m):
            if c[i] == j:
                nj = nj + 1
                means[j,:] = means[j,:] + X[i,:]
        if nj > 0:
            means[j,:] = means[j,:] / nj
    return means, c, np.sum(distortion)
```

Let's build models with $k \in 1..20$, plot the distortion for each k, and try to choose a good value for k using the so-called "elbow method."

In [37]:
```
# Convert dataframe to matrix

X = np.array(df.iloc[:,1:])

# Intialize hyperparameters

max_k = 20
epsilon = 0.001

# For each value of k, do one run and record the resulting cost
(Euclidean distortion)

distortions = np.zeros(max_k)
for k in range(1, max_k + 1):
    means = init_kmeans(X, k)
    prev_distortion = 0
    while True:
        means, c, distortion = iterate_kmeans(X, means)
        if prev_distortion > 0 and prev_distortion - distortion <
epsilon:
            break
        prev_distortion = distortion
    distortions[k-1] = distortion

# Plot distortion as function of k

plt.figure(figsize=(16,8))
plt.plot(range(1,max_k+1), distortions, 'bx-')
plt.xlabel('k')
plt.ylabel('Distortion')
plt.title('k-means distortion as a function of k')
plt.show()
```



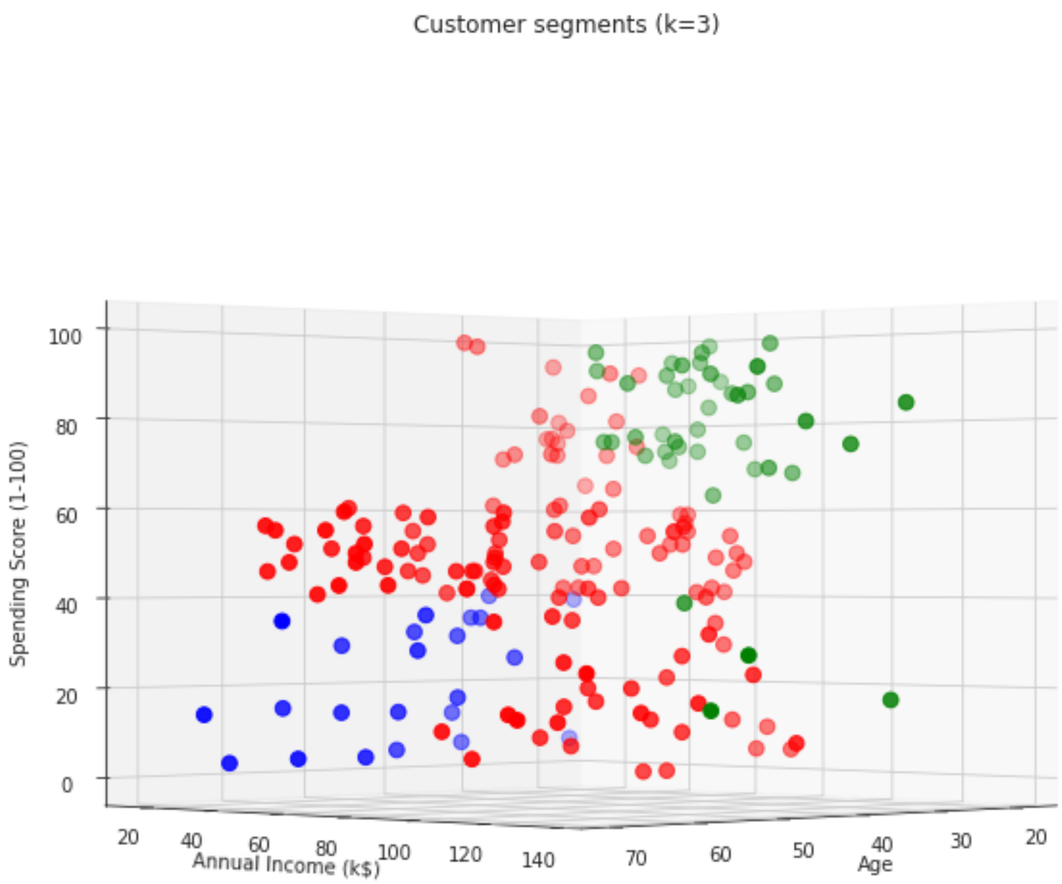
Read about the so-called "elbow method" in Wikipedia (https://en.wikipedia.org/wiki/Elbow_method_(clustering)). Note what it says, that "In practice there may not be a sharp elbow, and as a heuristic method, such an 'elbow' cannot always be unambiguously identified."

Do you see a unique elbow in the distortion plot above?

Note that the results are somewhat noisy, being dependent on initial conditions.

Here's a visualization of the results for three clusters:

In [38]:
```
# Re-run k-means with k=3

k = 3
means = init_kmeans(X, k)
prev_distortion = 0
while True:
    means, c, distortion = iterate_kmeans(X, means)
    if prev_distortion > 0 and prev_distortion - distortion < eps
ilon:
        break
    prev_distortion = distortion

# Set labels in dataset to cluster IDs according to k-means mode
l.

df["label"] = c

# Plot the data

fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(df.Age[df.label == 0], df["Annual Income (k$)"][df.lab
el == 0], df["Spending Score (1-100)"][df.label == 0], c='blue',
s=60)
ax.scatter(df.Age[df.label == 1], df["Annual Income (k$)"][df.lab
el == 1], df["Spending Score (1-100)"][df.label == 1], c='red', s
=60)
ax.scatter(df.Age[df.label == 2], df["Annual Income (k$)"][df.lab
el == 2], df["Spending Score (1-100)"][df.label == 2], c='green',
s=60)

# For 5 clusters, you can uncomment the following two lines.

#ax.scatter(df.Age[df.label == 3], df["Annual Income (k$)"][df.la
bel == 3], df["Spending Score (1-100)"][df.label == 3], c='orange
', s=60)
#ax.scatter(df.Age[df.label == 4], df["Annual Income (k$)"][df.la
bel == 4], df["Spending Score (1-100)"][df.label == 4], c='purple
', s=60)

ax.view_init(0, 45)
plt.xlabel("Age")
plt.ylabel("Annual Income (k$)")
ax.set_zlabel('Spending Score (1-100)')
plt.title('Customer segments (k=3)')
plt.show()
```



In-Lab Exercise 2

1. Consider the three cluster centers above. Look at the three means closely and come up with English descriptions of each cluster from a business point of view. Label the clusters in the visualization accordingly.
2. Note that the distortion plot is quite noisy due to random initial conditions. Modify the optimization to perfrom, for each k, several different runs, and take the minimum distortion over those runs. Re-plot the distortion plot and see if an "elbow" is more prominent.

Exercise 2.1 (10 points)

Consider the three cluster centers above. Look at the three means closely and come up with English descriptions of each cluster from a business point of view. Label the clusters in the visualization accordingly.

In [ ]:
```
# Your code here
```

Exercise 2.2 (20 points)

Note that the distortion plot is quite noisy due to random initial conditions. Modify the optimization to perfrom, for each k, several different runs, and take the minimum distortion over those runs. Re-plot the distortion plot and see if an "elbow" is more prominent.

In [ ]:
```
# Your code here
```

K-Means in PyTorch

Now, to get more experience with PyTorch, let's do the same thing with the library. First, some imports. You may need to install some packages for this to work:

```
pip install kmeans-pytorch
pip install tqdm
```

First, import the libraries:

In [39]:
```
!pip install kmeans-pytorch
!pip install tqdm
```

```
Collecting kmeans-pytorch
  Downloading kmeans_pytorch-0.3-py3-none-any.whl (4.4 kB)
Installing collected packages: kmeans-pytorch
Successfully installed kmeans-pytorch-0.3
Requirement already satisfied: tqdm in /home/alisa/anaconda3/lib/
python3.8/site-packages (4.47.0)
```

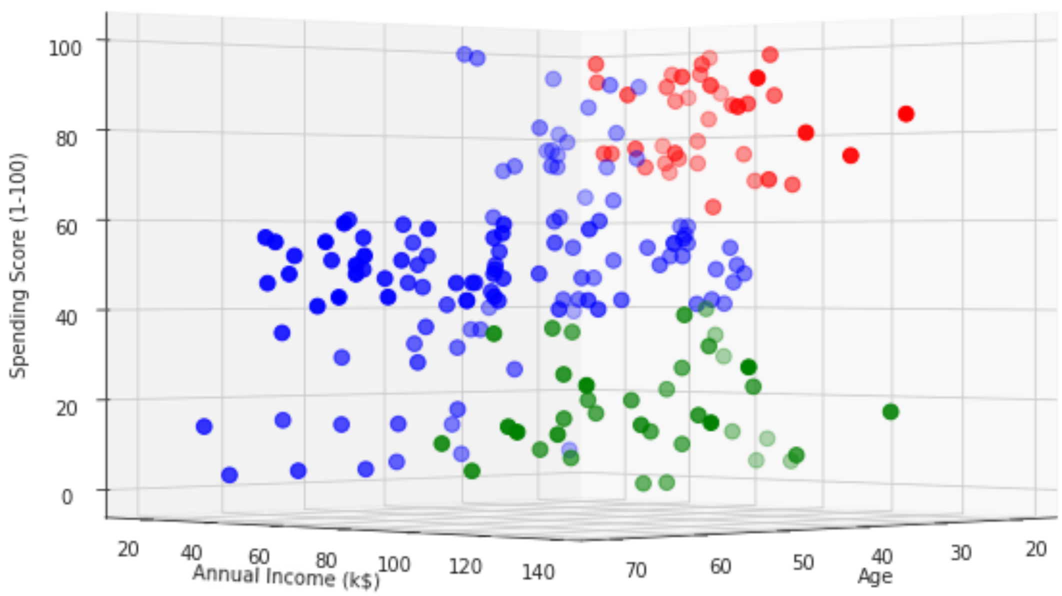
In [40]:
```
import torch
from kmeans_pytorch import kmeans
```

In [41]:
```
x =  torch.from_numpy(X)
device = 'cuda:0'
device = 'cpu'
c, means = kmeans(X=x, num_clusters=3, distance='euclidean', devi
ce=torch.device(device))
df["label"] = c
```

```
[running kmeans]: 15it [00:00, 889.87it/s, center_shift=0.000000,
iteration=15, tol=0.000100]

running k-means on cpu..
```

In [42]:
```
fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(df.Age[df.label == 0], df["Annual Income (k$)"][df.label == 0], df["Spending Score (1-100)"][df.label == 0], c='blue', s=60)
ax.scatter(df.Age[df.label == 1], df["Annual Income (k$)"][df.label == 1], df["Spending Score (1-100)"][df.label == 1], c='red', s=60)
ax.scatter(df.Age[df.label == 2], df["Annual Income (k$)"][df.label == 2], df["Spending Score (1-100)"][df.label == 2], c='green', s=60)
#ax.scatter(df.Age[df.label == 3], df["Annual Income (k$)"][df.label == 3], df["Spending Score (1-100)"][df.label == 3], c='orange', s=60)
#ax.scatter(df.Age[df.label == 4], df["Annual Income (k$)"][df.label == 4], df["Spending Score (1-100)"][df.label == 4], c='purple', s=60)
ax.view_init(0, 45)
plt.xlabel("Age")
plt.ylabel("Annual Income (k$)")
ax.set_zlabel('Spending Score (1-100)')
plt.title('Customer Segments (PyTorch k=3)')
plt.show()
```

Customer Segments (PyTorch k=3)



Take-Home Exercise

Find an interesting dataset for unsupervised learning, prepare the data, and run k-means on it.

In a brief report, describe your in-lab and take home experiments and their results.

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```