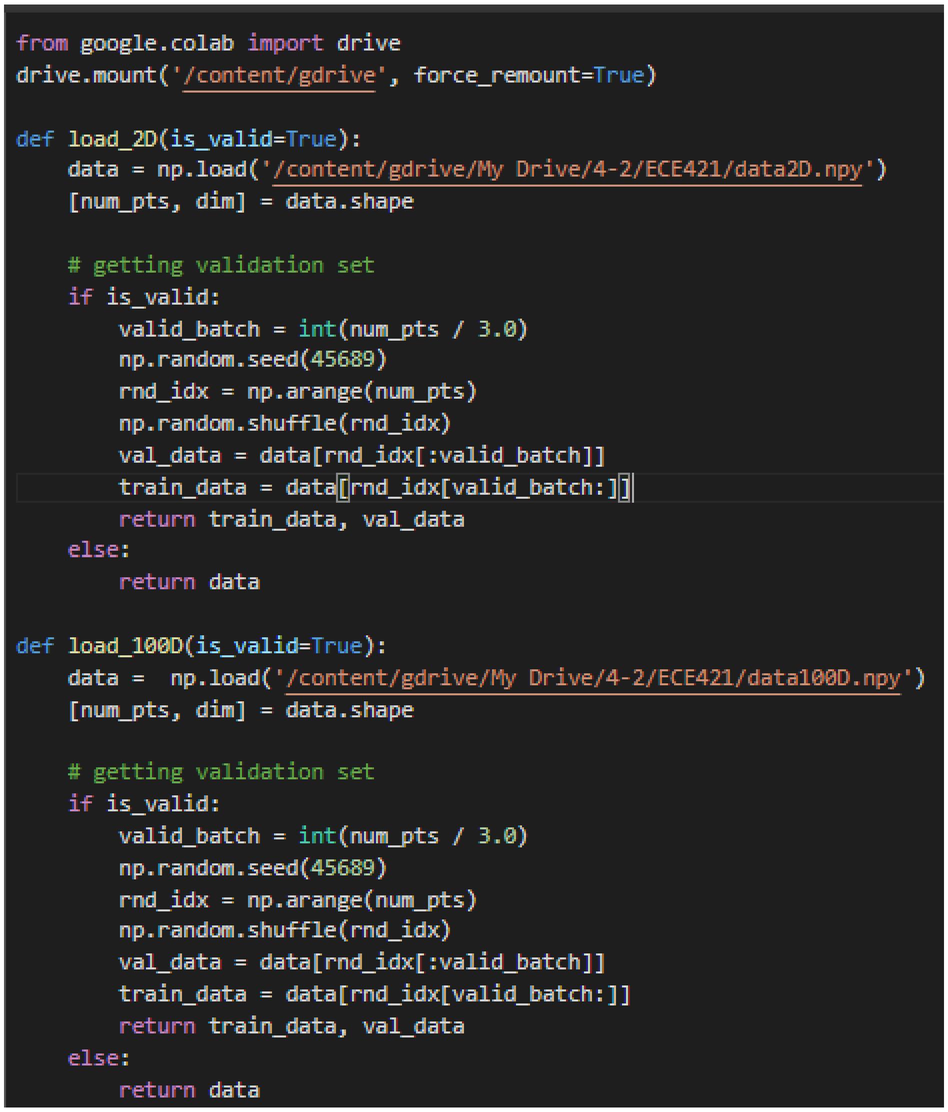
ECE421 - Winter 2022 Assignment 3:
Unsupervised Learning and Probabilistic Models

Raymond Hong 1003942629

Code written in Google Colab: https://colab.research.google.com/drive/1TOGHI-snK5q5YR9Hac1PCcwV8NNDjeQ0

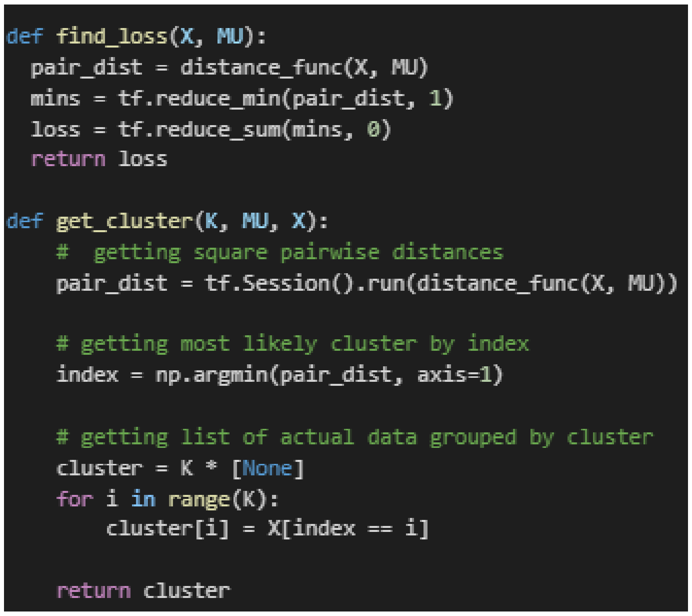
0. Helper Functions

0.1 Loading Data



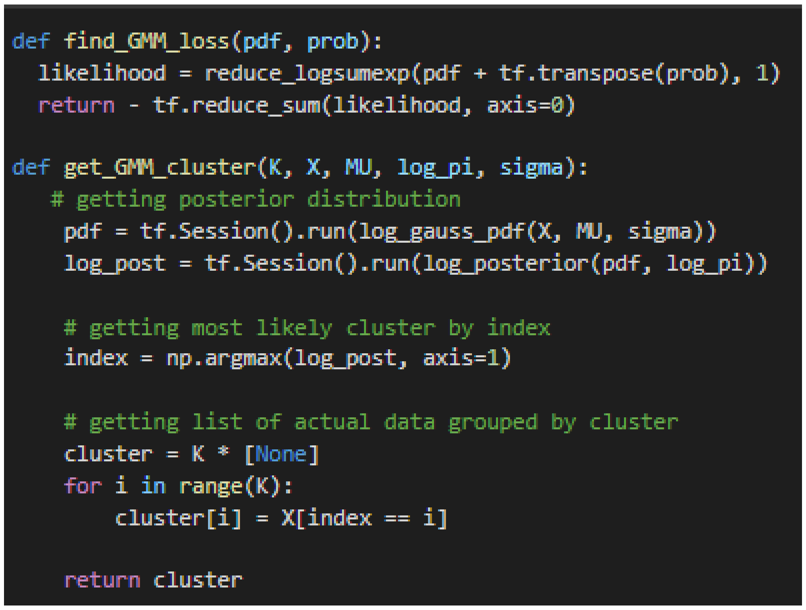
```
from google.colab import drive
drive.mount('/content/gdrive', force_remount=True)

def load_2D(is_valid=True):
    data = np.load('/content/gdrive/My Drive/4-2/ECE421/data2D.npy')
    [num_pts, dim] = data.shape

    # getting validation set
    if is_valid:
        valid_batch = int(num_pts / 3.0)
        np.random.seed(45689)
        rnd_idx = np.arange(num_pts)
        np.random.shuffle(rnd_idx)
        val_data = data[rnd_idx[:valid_batch]]
        train_data = data[rnd_idx[valid_batch:]]
        return train_data, val_data
    else:
        return data

def load_100D(is_valid=True):
    data =  np.load('/content/gdrive/My Drive/4-2/ECE421/data100D.npy')
    [num_pts, dim] = data.shape

    # getting validation set
    if is_valid:
        valid_batch = int(num_pts / 3.0)
        np.random.seed(45689)
        rnd_idx = np.arange(num_pts)
        np.random.shuffle(rnd_idx)
        val_data = data[rnd_idx[:valid_batch]]
        train_data = data[rnd_idx[valid_batch:]]
        return train_data, val_data
    else:
        return data
```

0.2 K-means Helper Functions



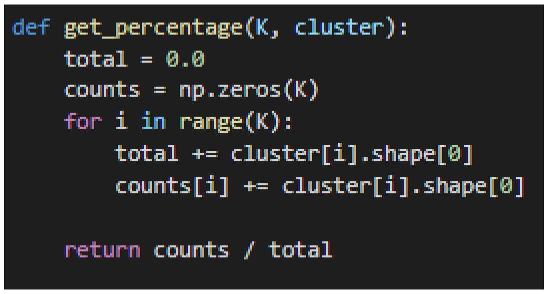
```python
def find_loss(X, MU):
  pair_dist = distance_func(X, MU)
  mins = tf.reduce_min(pair_dist, 1)
  loss = tf.reduce_sum(mins, 0)
  return loss

def get_cluster(K, MU, X):
    #  getting square pairwise distances
    pair_dist = tf.Session().run(distance_func(X, MU))

    # getting most likely cluster by index
    index = np.argmin(pair_dist, axis=1)

    # getting list of actual data grouped by cluster
    cluster = K * [None]
    for i in range(K):
        cluster[i] = X[index == i]

    return cluster
```

0.3 GMM Helper Functions



```python
def find_GMM_loss(pdf, prob):
  likelihood = reduce_logsumexp(pdf + tf.transpose(prob), 1)
  return - tf.reduce_sum(likelihood, axis=0)

def get_GMM_cluster(K, X, MU, log_pi, sigma):
   # getting posterior distribution
    pdf = tf.Session().run(log_gauss_pdf(X, MU, sigma))
    log_post = tf.Session().run(log_posterior(pdf, log_pi))

    # getting most likely cluster by index
    index = np.argmax(log_post, axis=1)

    # getting list of actual data grouped by cluster
    cluster = K * [None]
    for i in range(K):
        cluster[i] = X[index == i]

    return cluster
```

0.4 Getting Percentages



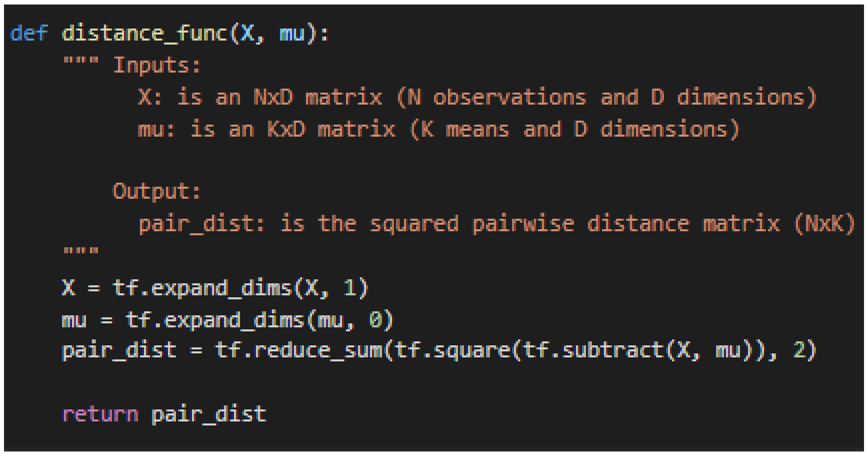
```python
def get_percentage(K, cluster):
    total = 0.0
    counts = np.zeros(K)
    for i in range(K):
        total += cluster[i].shape[0]
        counts[i] += cluster[i].shape[0]

    return counts / total
```

1. K-Means

1.1 Learning K-means

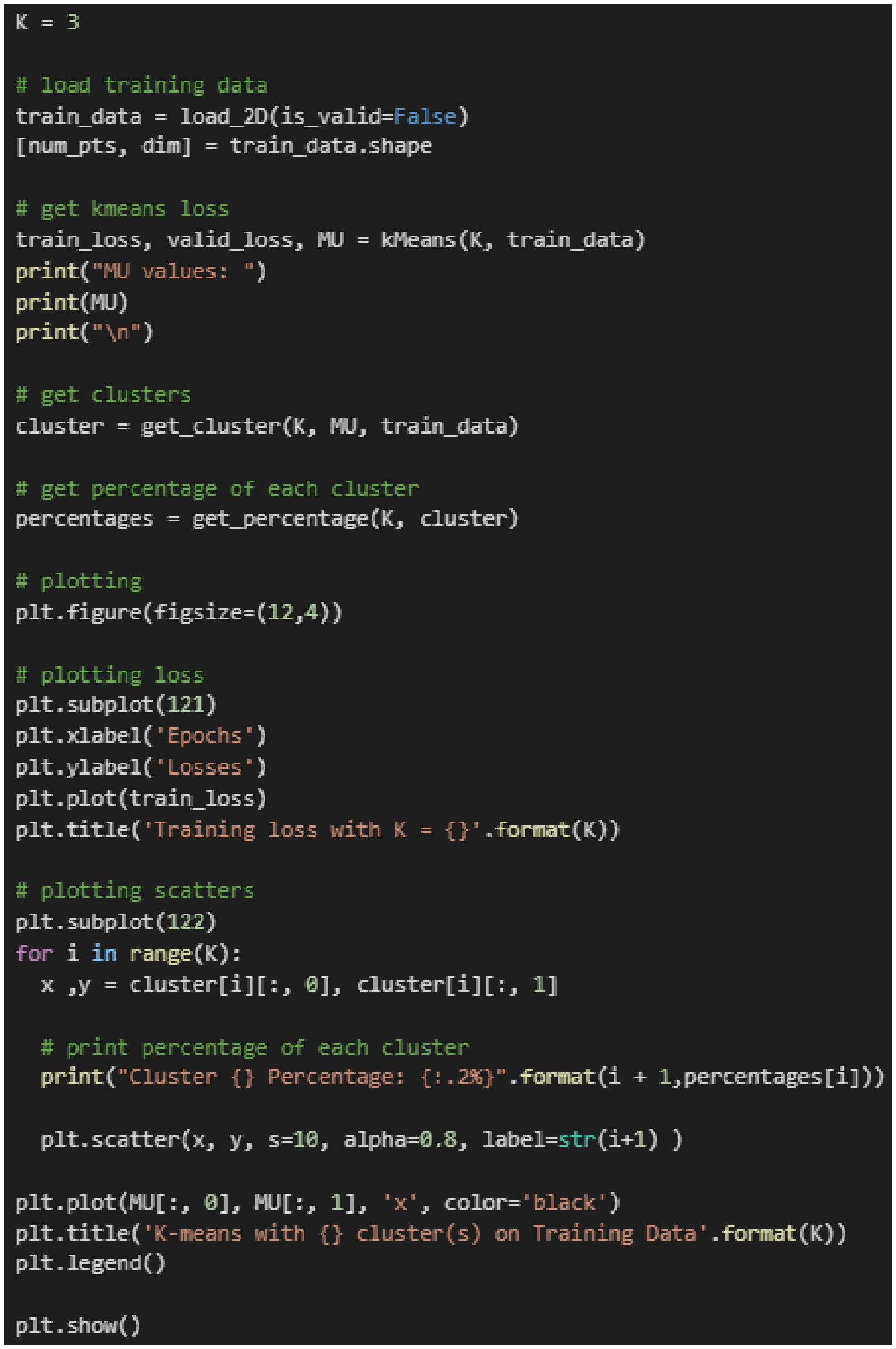
1.1.1 Model Implementation

Implemented function **distance_func()**

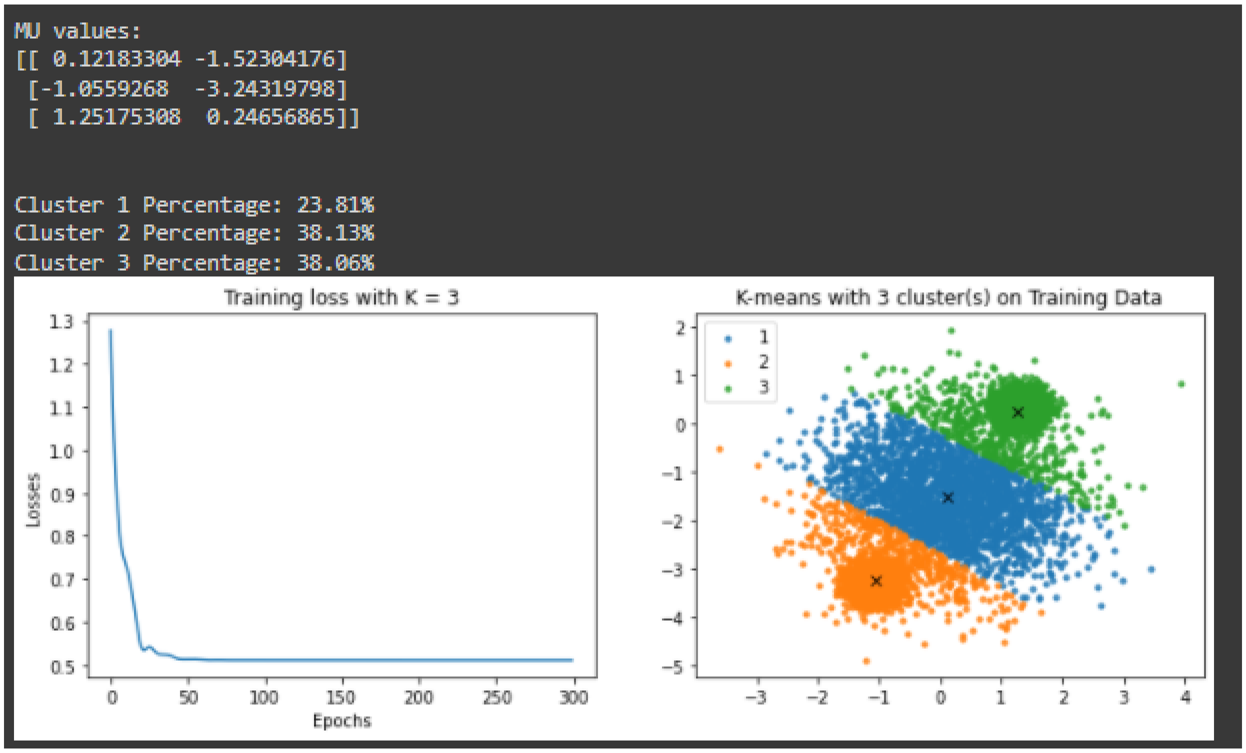


```
def distance_func(X, mu):
    """ Inputs:
          X: is an NxD matrix (N observations and D dimensions)
          mu: is an KxD matrix (K means and D dimensions)

        Output:
          pair_dist: is the squared pairwise distance matrix (NxK)
    """
    X = tf.expand_dims(X, 1)
    mu = tf.expand_dims(mu, 0)
    pair_dist = tf.reduce_sum(tf.square(tf.subtract(X, mu)), 2)

    return pair_dist
```

1.1.2 K-means Clusters when K = 3



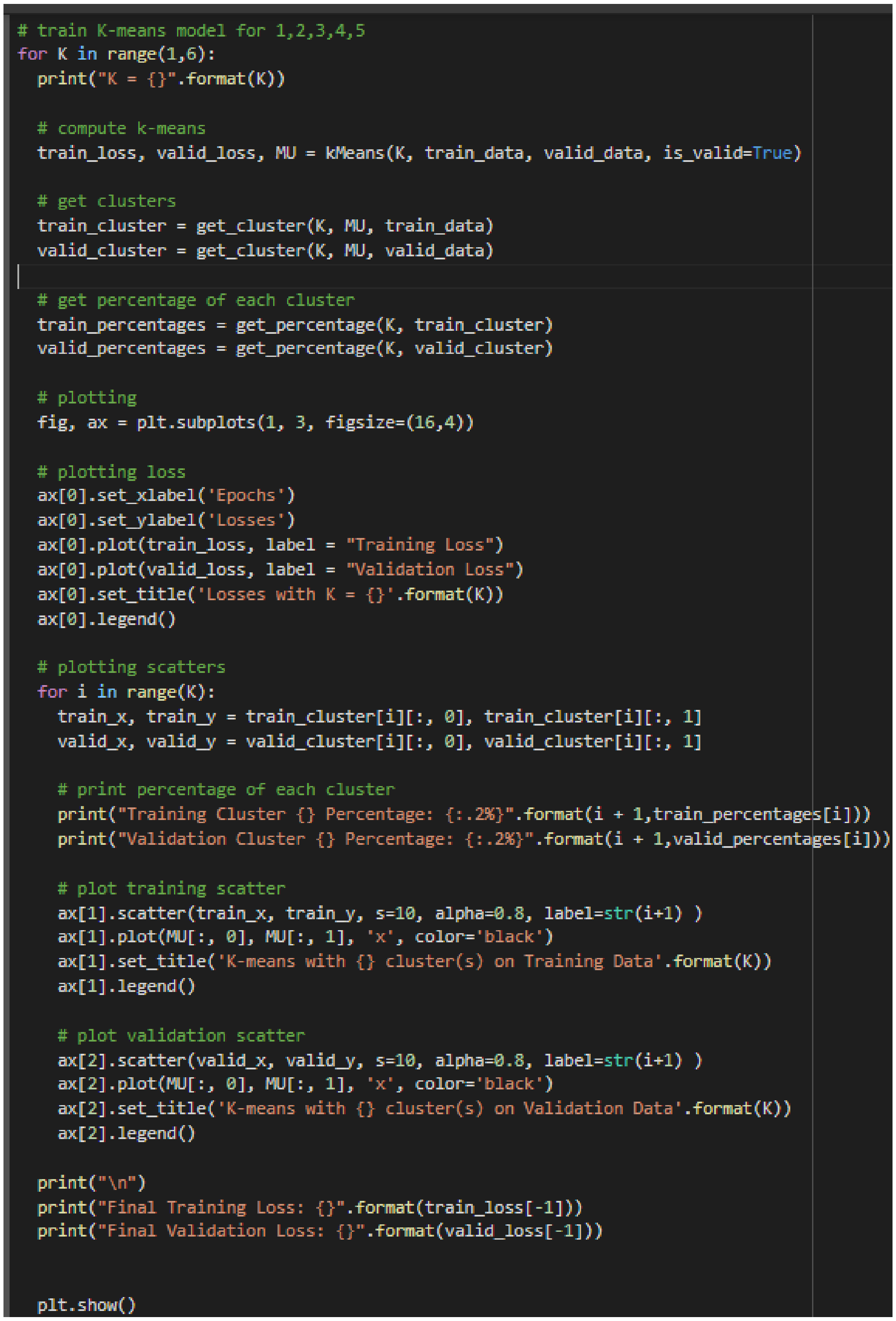
```python
K = 3

# load training data
train_data = load_2D(is_valid=False)
[num_pts, dim] = train_data.shape

# get kmeans loss
train_loss, valid_loss, MU = kMeans(K, train_data)
print("MU values: ")
print(MU)
print("\n")

# get clusters
cluster = get_cluster(K, MU, train_data)

# get percentage of each cluster
percentages = get_percentage(K, cluster)

# plotting
plt.figure(figsize=(12,4))

# plotting loss
plt.subplot(121)
plt.xlabel('Epochs')
plt.ylabel('Losses')
plt.plot(train_loss)
plt.title('Training loss with K = {}'.format(K))

# plotting scatters
plt.subplot(122)
for i in range(K):
  x ,y = cluster[i][:, 0], cluster[i][:, 1]

  # print percentage of each cluster
  print("Cluster {} Percentage: {:.2%}".format(i + 1,percentages[i]))

  plt.scatter(x, y, s=10, alpha=0.8, label=str(i+1) )

plt.plot(MU[:, 0], MU[:, 1], 'x', color='black')
plt.title('K-means with {} cluster(s) on Training Data'.format(K))
plt.legend()

plt.show()
```



```
MU values:
[[ 0.12183304 -1.52304176]
 [-1.0559268  -3.24319798]
 [ 1.25175308  0.24656865]]


Cluster 1 Percentage: 23.81%
Cluster 2 Percentage: 38.13%
Cluster 3 Percentage: 38.06%
```

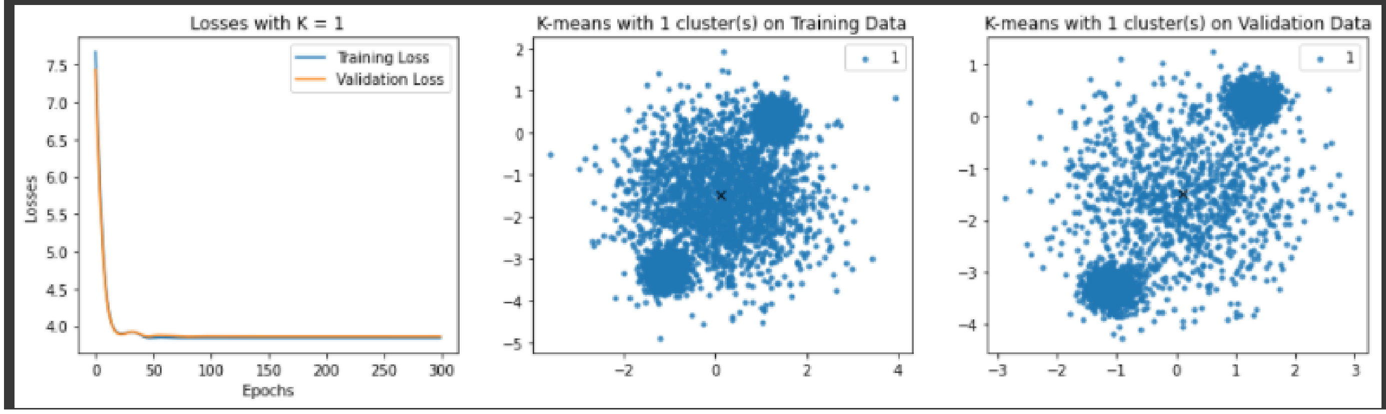
1.1.3 Varying K with Validation

Trained the K-means model for K = 1,2,3,4,5.

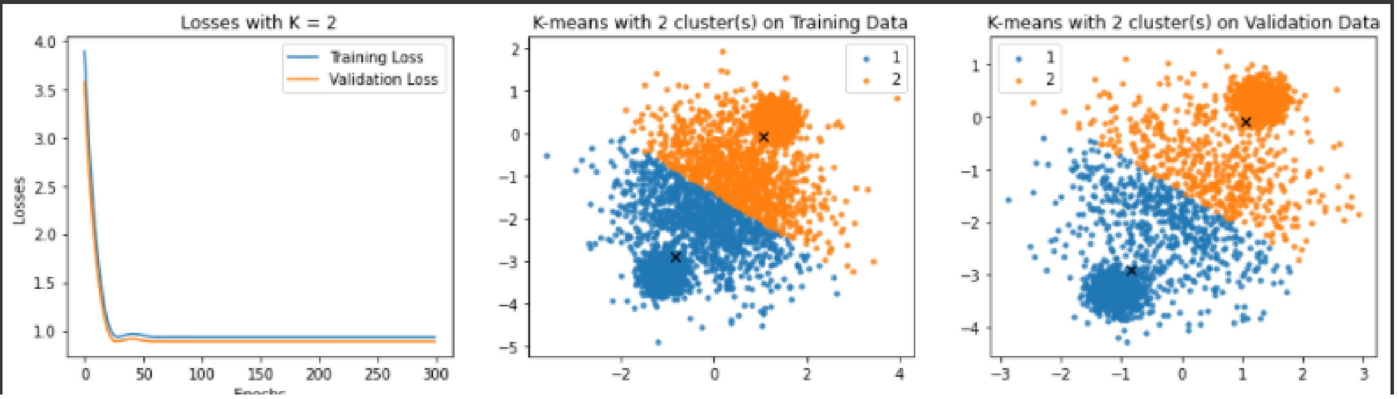


```
# train K-means model for 1,2,3,4,5
for K in range(1,6):
  print("K = {}".format(K))

  # compute k-means
  train_loss, valid_loss, MU = kMeans(K, train_data, valid_data, is_valid=True)

  # get clusters
  train_cluster = get_cluster(K, MU, train_data)
  valid_cluster = get_cluster(K, MU, valid_data)

  # get percentage of each cluster
  train_percentages = get_percentage(K, train_cluster)
  valid_percentages = get_percentage(K, valid_cluster)

  # plotting
  fig, ax = plt.subplots(1, 3, figsize=(16,4))

  # plotting loss
  ax[0].set_xlabel('Epochs')
  ax[0].set_ylabel('Losses')
  ax[0].plot(train_loss, label = "Training Loss")
  ax[0].plot(valid_loss, label = "Validation Loss")
  ax[0].set_title('Losses with K = {}'.format(K))
  ax[0].legend()

  # plotting scatters
  for i in range(K):
    train_x, train_y = train_cluster[i][:, 0], train_cluster[i][:, 1]
    valid_x, valid_y = valid_cluster[i][:, 0], valid_cluster[i][:, 1]

    # print percentage of each cluster
    print("Training Cluster {} Percentage: {:.2%}".format(i + 1,train_percentages[i]))
    print("Validation Cluster {} Percentage: {:.2%}".format(i + 1,valid_percentages[i]))

    # plot training scatter
    ax[1].scatter(train_x, train_y, s=10, alpha=0.8, label=str(i+1) )
    ax[1].plot(MU[:, 0], MU[:, 1], 'x', color='black')
    ax[1].set_title('K-means with {} cluster(s) on Training Data'.format(K))
    ax[1].legend()

    # plot validation scatter
    ax[2].scatter(valid_x, valid_y, s=10, alpha=0.8, label=str(i+1) )
    ax[2].plot(MU[:, 0], MU[:, 1], 'x', color='black')
    ax[2].set_title('K-means with {} cluster(s) on Validation Data'.format(K))
    ax[2].legend()

  print("\n")
  print("Final Training Loss: {}".format(train_loss[-1]))
  print("Final Validation Loss: {}".format(valid_loss[-1]))


  plt.show()
```

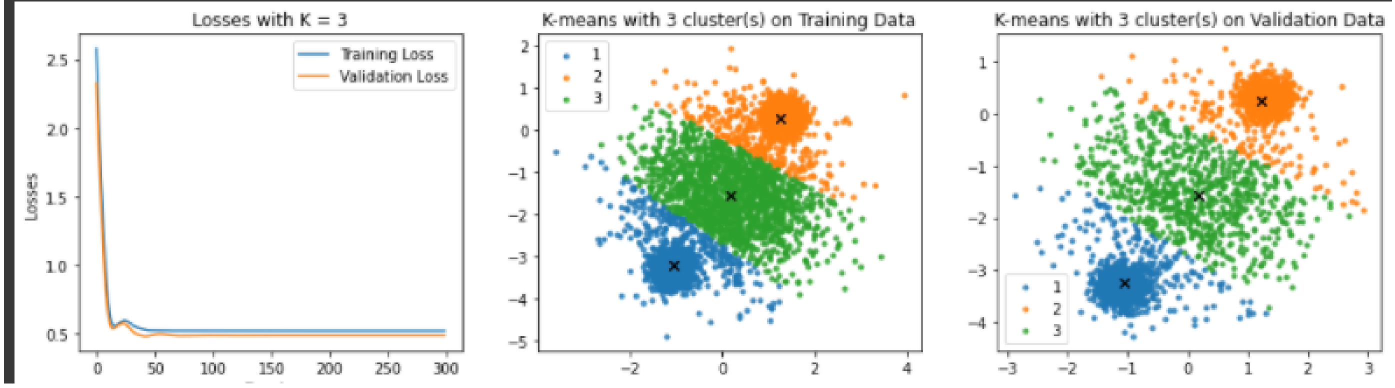
```
K = 1
Training Cluster 1 Percentage: 100.00%
Validation Cluster 1 Percentage: 100.00%


Final Training Loss: 3.8381579392668326
Final Validation Loss: 3.861417256672835
```



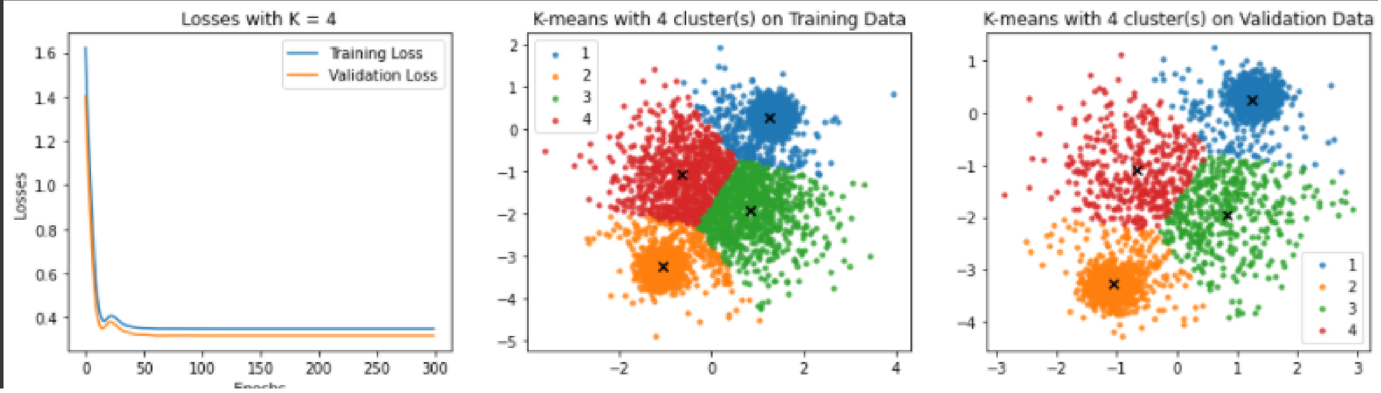
```
K = 2
Training Cluster 1 Percentage: 49.81%
Validation Cluster 1 Percentage: 51.82%
Training Cluster 2 Percentage: 50.19%
Validation Cluster 2 Percentage: 48.18%


Final Training Loss: 0.9364531984448655
Final Validation Loss: 0.8882906883936502
```



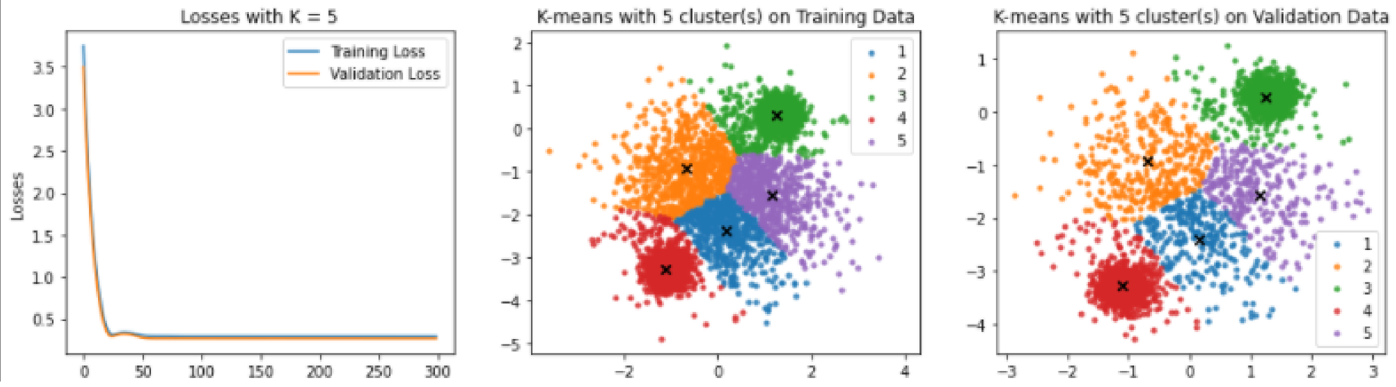
```
K = 3
Training Cluster 1 Percentage: 37.51%
Validation Cluster 1 Percentage: 39.78%
Training Cluster 2 Percentage: 38.41%
Validation Cluster 2 Percentage: 37.35%
Training Cluster 3 Percentage: 24.07%
Validation Cluster 3 Percentage: 22.86%


Final Training Loss: 0.5233501591809401
Final Validation Loss: 0.488841713259277
```



```
K = 4
Training Cluster 1 Percentage: 37.63%
Validation Cluster 1 Percentage: 36.57%
Training Cluster 2 Percentage: 36.33%
Validation Cluster 2 Percentage: 38.79%
Training Cluster 3 Percentage: 13.72%
Validation Cluster 3 Percentage: 12.66%
Training Cluster 4 Percentage: 12.31%
Validation Cluster 4 Percentage: 11.97%


Final Training Loss: 0.3480055727294938
Final Validation Loss: 0.3163930144990918
```



```
K = 5
Training Cluster 1 Percentage: 8.83%
Validation Cluster 1 Percentage: 8.31%
Training Cluster 2 Percentage: 10.68%
Validation Cluster 2 Percentage: 10.68%
Training Cluster 3 Percentage: 36.70%
Validation Cluster 3 Percentage: 35.76%
Training Cluster 4 Percentage: 34.89%
Validation Cluster 4 Percentage: 37.38%
Training Cluster 5 Percentage: 8.89%
Validation Cluster 5 Percentage: 7.86%


Final Training Loss: 0.2940552930347517
Final Validation Loss: 0.26672073221951187
```



The resulting percentages and final losses for each K is demonstrated here:

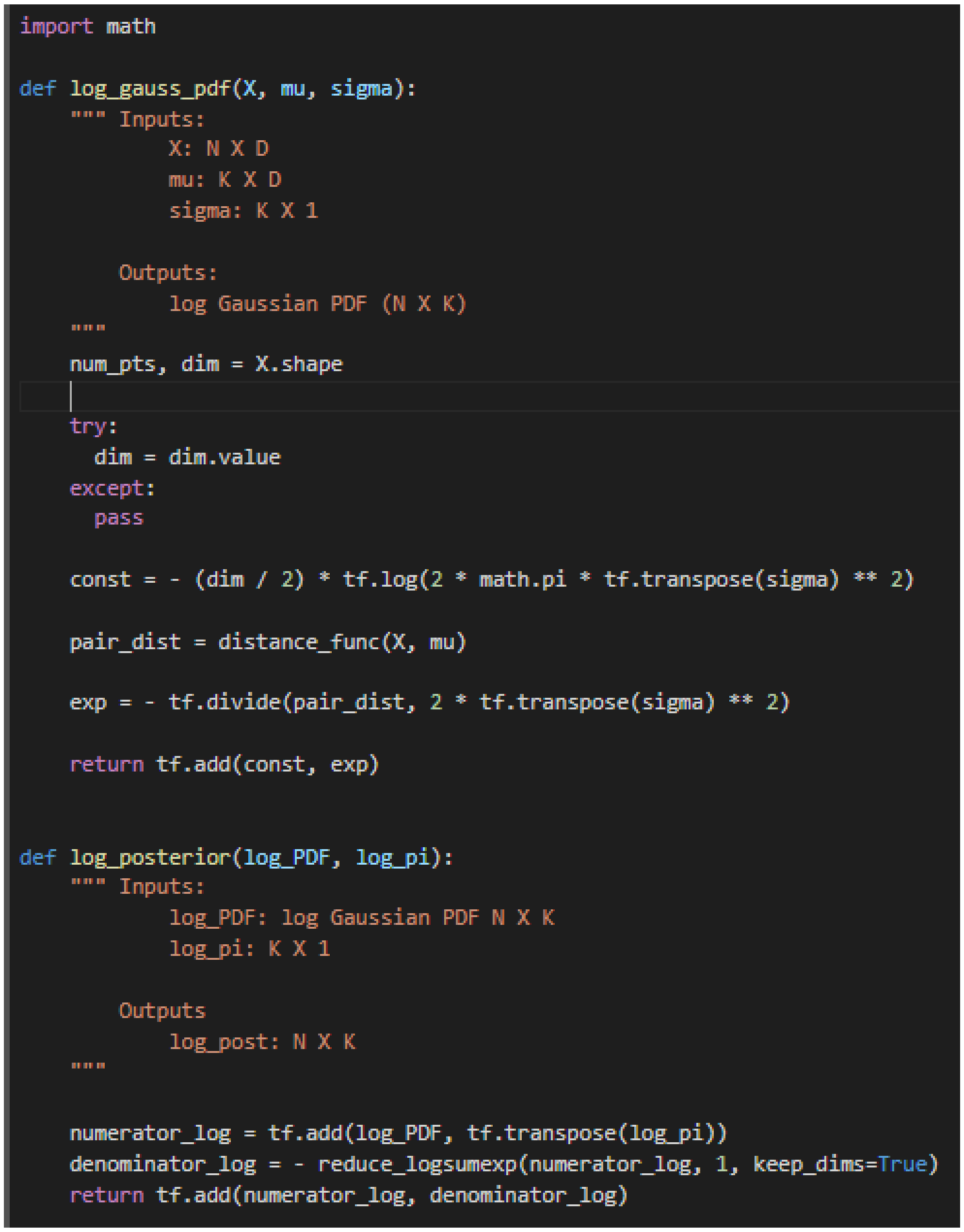
K	1	2	3	4	5
Final Training Loss	3.8381579392668326	0.9364531984448655	0.5233501591809401	0.3480055727294938	0.2940552930347517
Final Validation Loss	3.861417256672835	0.8882906883936502	0.488841713259277	0.3163930144990918	0.26672073221951187
Training Cluster Percentage	Cluster 1: 100 %	Cluster 1: 49.81% Cluster 2: 50.19%	Cluster 1: 37.51% Cluster 2: 38.41% Cluster 3: 24.07%	Cluster 1: 37.63% Cluster 2: 36.33% Cluster 3: 13.72% Cluster 4: 12.31%	Cluster 1: 8.83% Cluster 2: 10.68% Cluster 3: 36.70% Cluster 4: 34.89% Cluster 5: 8.89%
Validation Cluster Percentage	Cluster 1: 100%	Cluster 1: 51.82% Cluster 2: 48.18%	Cluster 1: 39.78% Cluster 2: 37.35% Cluster 3: 22.86%	Cluster 1: 36.57% Cluster 2: 38.79% Cluster 3: 12.66% Cluster 4: 11.97%	Cluster 1: 8.31% Cluster 2: 10.68% Cluster 3: 35.76% Cluster 4: 37.38% Cluster 5: 7.86%

Discussion

The best number of clusters to use is **3**. There are two highly populated areas on the top right and left bottom corners of the scatter plots. Other than these two areas, data scatters across the board. When K > 3, the 2 areas on the corners stay, and only the middle part splits into smaller parts. This is not too accurate as the scattered area in the middle should not have multiple centers. In addition, when K < 3, the scatter plot fails to capture the significant difference between the 3 parts. Therefore, 3 is the best number of clusters to use.

2. Mixtures of Gaussians

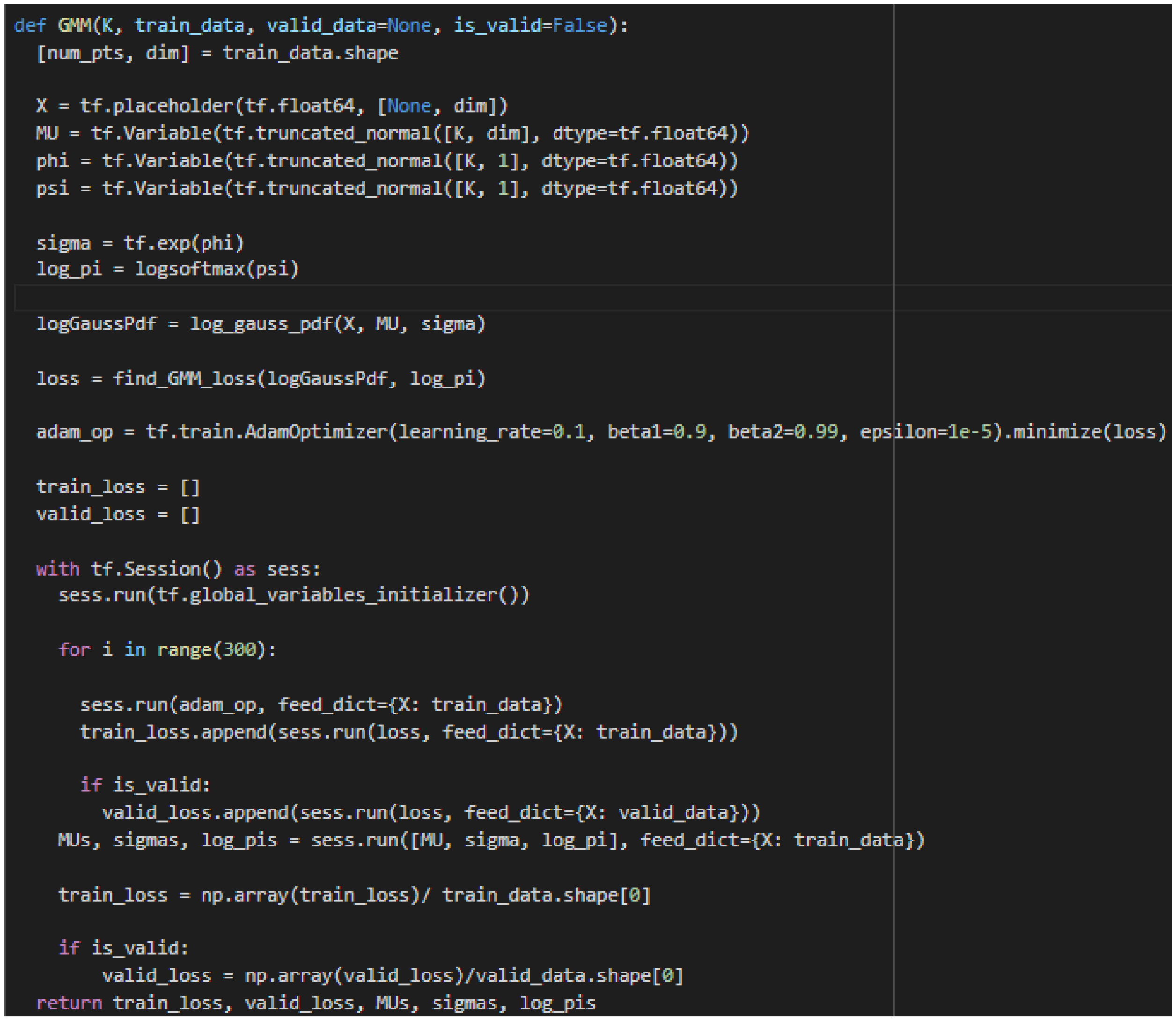
2.1 The Gaussian Cluster Mode



```python
import math

def log_gauss_pdf(X, mu, sigma):
    """ Inputs:
            X: N X D
            mu: K X D
            sigma: K X 1

        Outputs:
            log Gaussian PDF (N X K)
    """
    num_pts, dim = X.shape

    try:
      dim = dim.value
    except:
      pass

    const = - (dim / 2) * tf.log(2 * math.pi * tf.transpose(sigma) ** 2)

    pair_dist = distance_func(X, mu)

    exp = - tf.divide(pair_dist, 2 * tf.transpose(sigma) ** 2)

    return tf.add(const, exp)


def log_posterior(log_PDF, log_pi):
    """ Inputs:
            log_PDF: log Gaussian PDF N X K
            log_pi: K X 1

        Outputs
            log_post: N X K
    """

    numerator_log = tf.add(log_PDF, tf.transpose(log_pi))
    denominator_log = - reduce_logsumexp(numerator_log, 1, keep_dims=True)
    return tf.add(numerator_log, denominator_log)
```

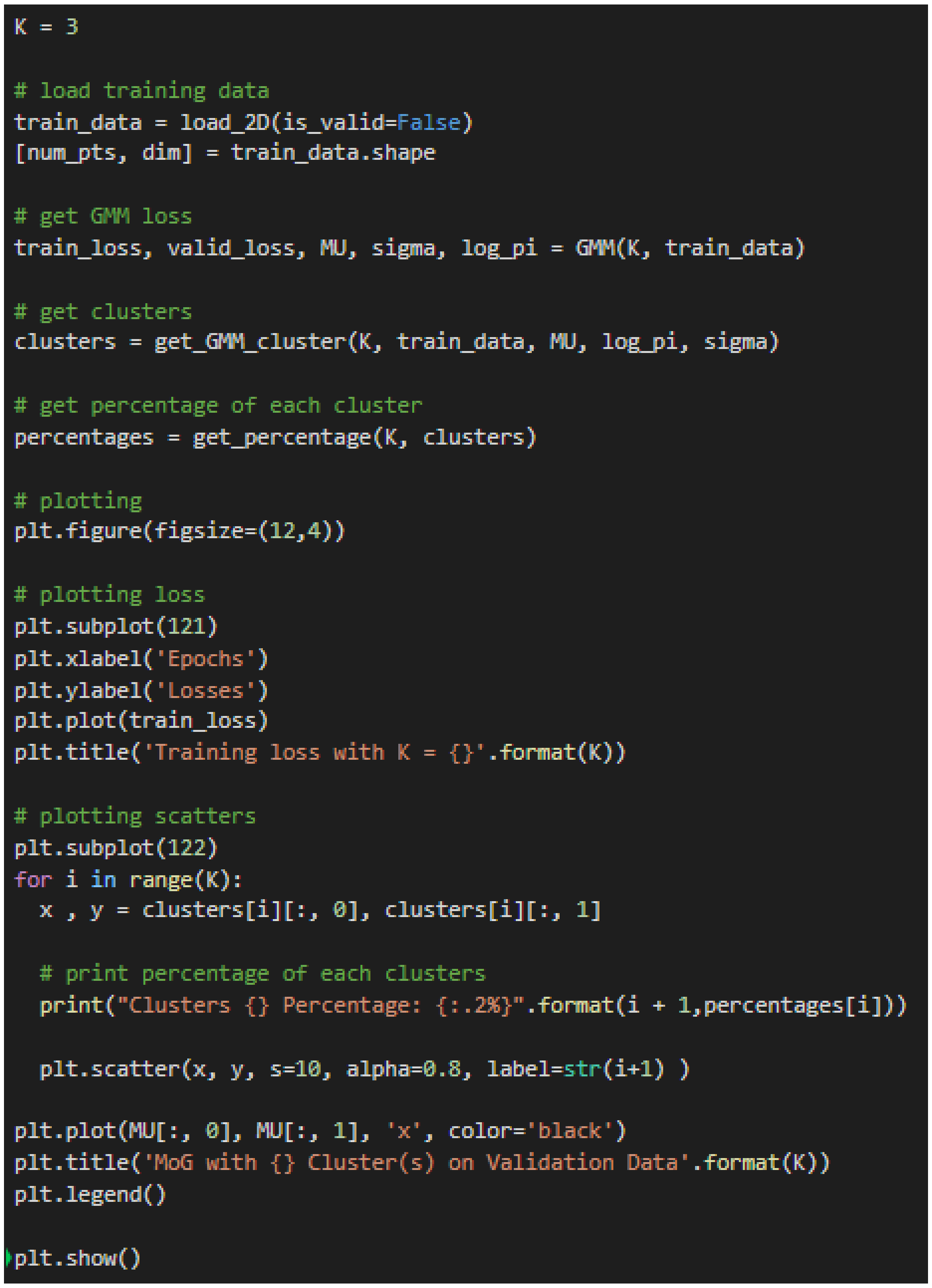
2.2 Learning the MoG

2.2.1 Model Implementation

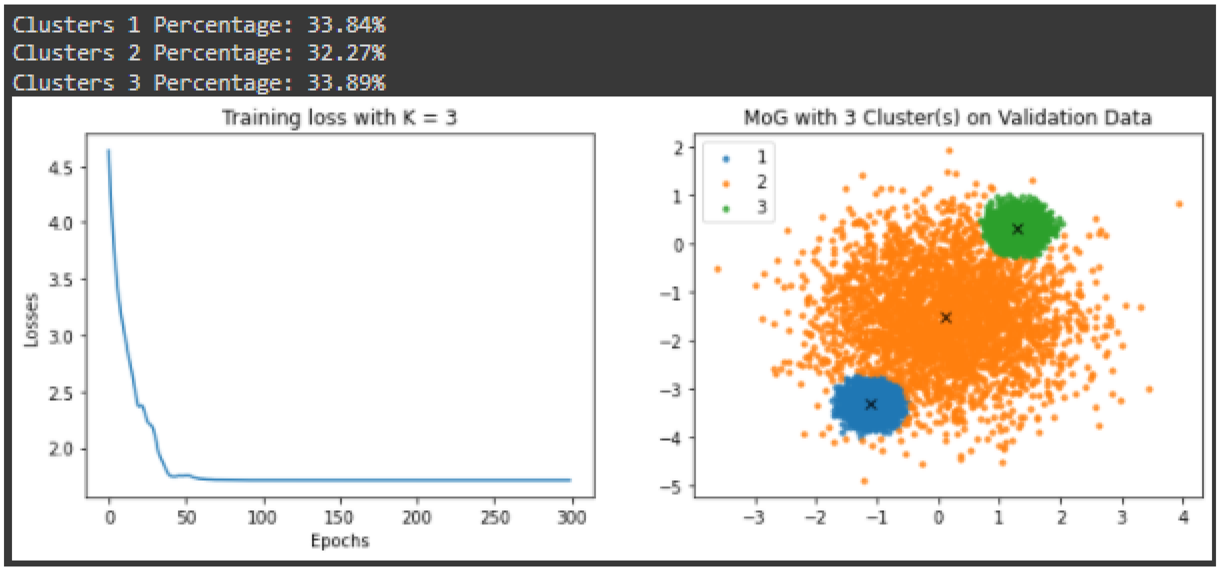


```
def GMM(K, train_data, valid_data=None, is_valid=False):
  [num_pts, dim] = train_data.shape

  X = tf.placeholder(tf.float64, [None, dim])
  MU = tf.Variable(tf.truncated_normal([K, dim], dtype=tf.float64))
  phi = tf.Variable(tf.truncated_normal([K, 1], dtype=tf.float64))
  psi = tf.Variable(tf.truncated_normal([K, 1], dtype=tf.float64))

  sigma = tf.exp(phi)
  log_pi = logsoftmax(psi)

  logGaussPdf = log_gauss_pdf(X, MU, sigma)

  loss = find_GMM_loss(logGaussPdf, log_pi)

  adam_op = tf.train.AdamOptimizer(learning_rate=0.1, beta1=0.9, beta2=0.99, epsilon=1e-5).minimize(loss)

  train_loss = []
  valid_loss = []

  with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for i in range(300):

      sess.run(adam_op, feed_dict={X: train_data})
      train_loss.append(sess.run(loss, feed_dict={X: train_data}))

      if is_valid:
        valid_loss.append(sess.run(loss, feed_dict={X: valid_data}))
    MUs, sigmas, log_pis = sess.run([MU, sigma, log_pi], feed_dict={X: train_data})

    train_loss = np.array(train_loss)/ train_data.shape[0]

    if is_valid:
        valid_loss = np.array(valid_loss)/valid_data.shape[0]
  return train_loss, valid_loss, MUs, sigmas, log_pis
```

2.2.2 MoG Clusters with K = 3



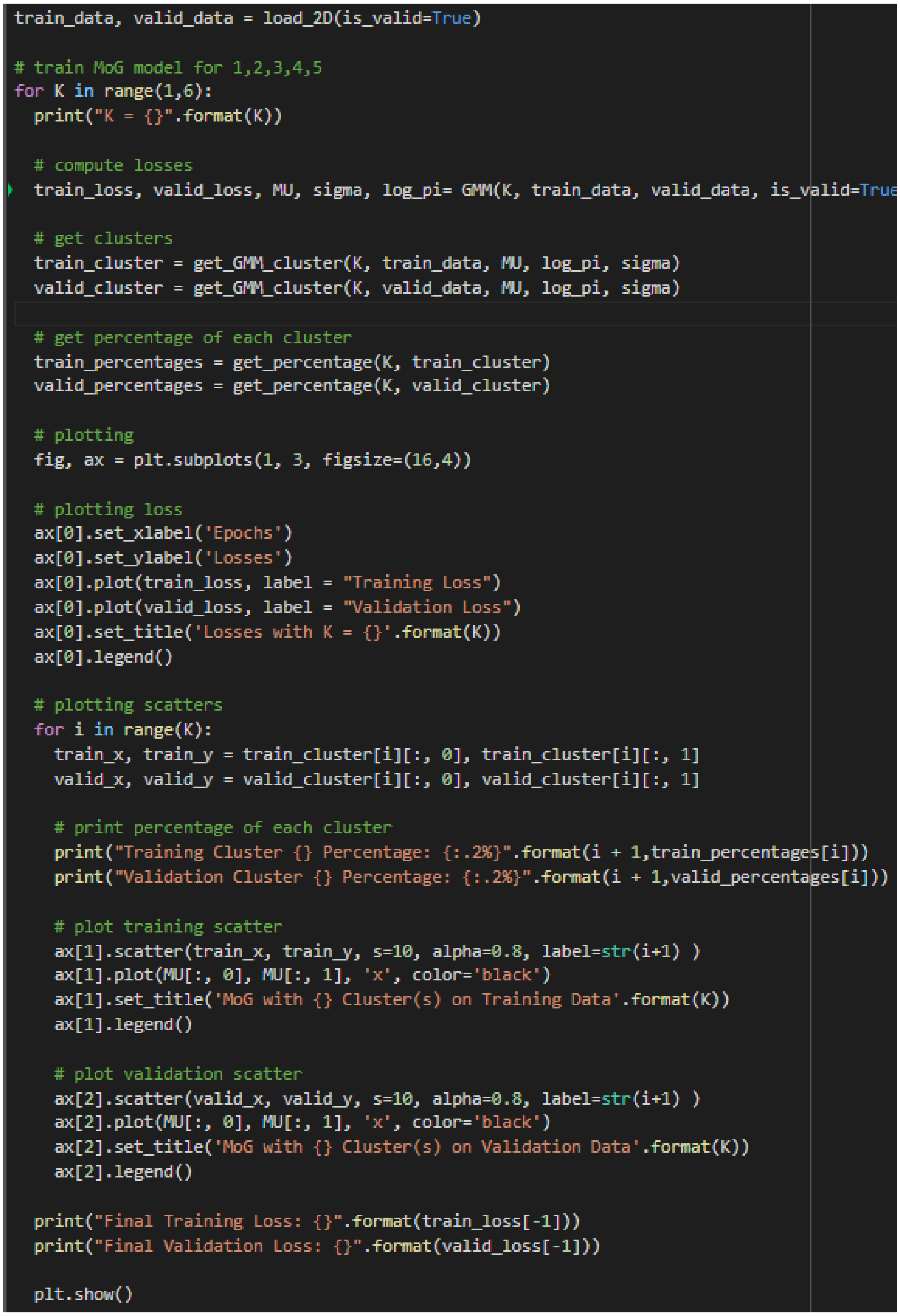
```python
K = 3

# load training data
train_data = load_2D(is_valid=False)
[num_pts, dim] = train_data.shape

# get GMM loss
train_loss, valid_loss, MU, sigma, log_pi = GMM(K, train_data)

# get clusters
clusters = get_GMM_cluster(K, train_data, MU, log_pi, sigma)

# get percentage of each cluster
percentages = get_percentage(K, clusters)

# plotting
plt.figure(figsize=(12,4))

# plotting loss
plt.subplot(121)
plt.xlabel('Epochs')
plt.ylabel('Losses')
plt.plot(train_loss)
plt.title('Training loss with K = {}'.format(K))

# plotting scatters
plt.subplot(122)
for i in range(K):
  x , y = clusters[i][:, 0], clusters[i][:, 1]

  # print percentage of each clusters
  print("Clusters {} Percentage: {:.2%}".format(i + 1,percentages[i]))

  plt.scatter(x, y, s=10, alpha=0.8, label=str(i+1) )

plt.plot(MU[:, 0], MU[:, 1], 'x', color='black')
plt.title('MoG with {} Cluster(s) on Validation Data'.format(K))
plt.legend()

plt.show()
```



```
Clusters 1 Percentage: 33.84%
Clusters 2 Percentage: 32.27%
Clusters 3 Percentage: 33.89%
```

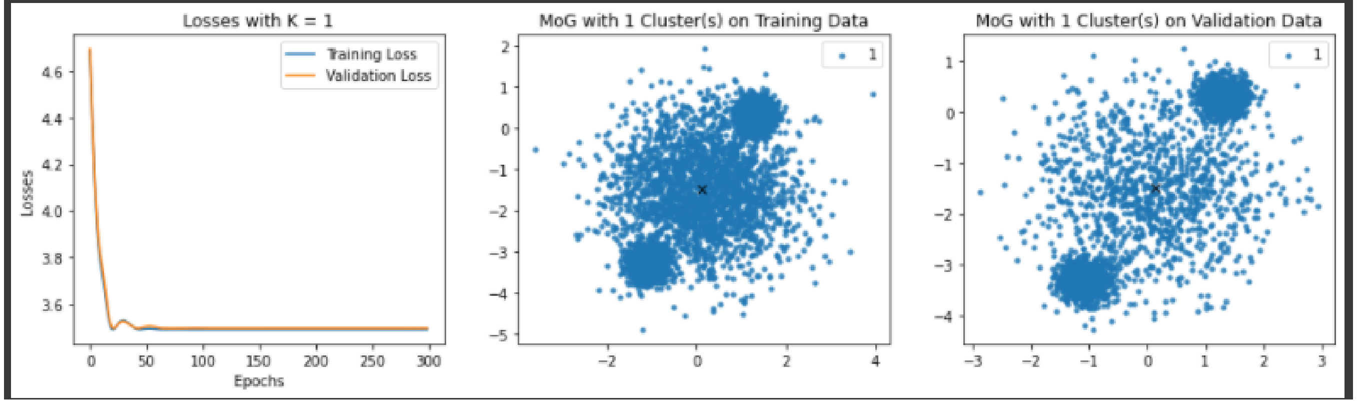
2.2.3 Varying K with Validation

Trained the MoG model for K = 1,2,3,4,5.

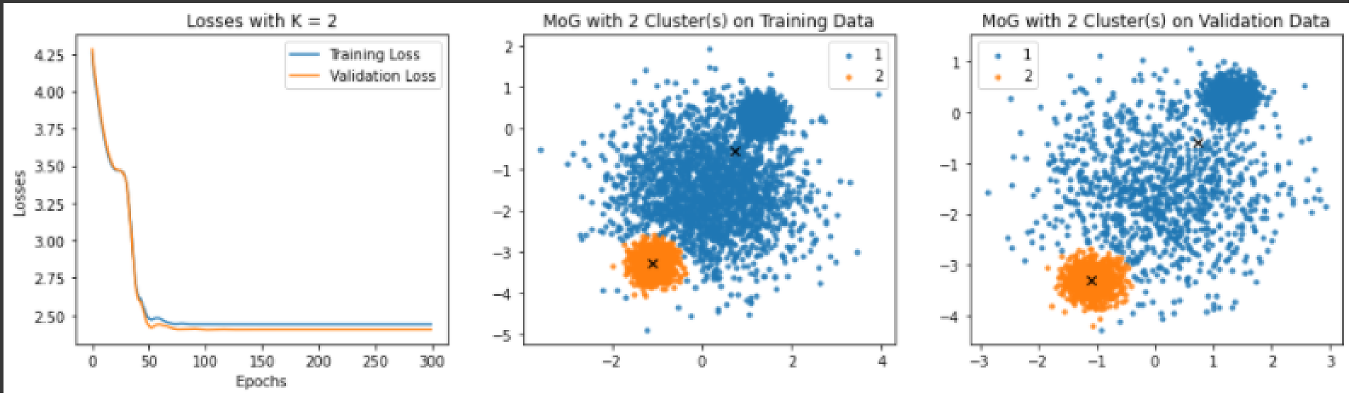


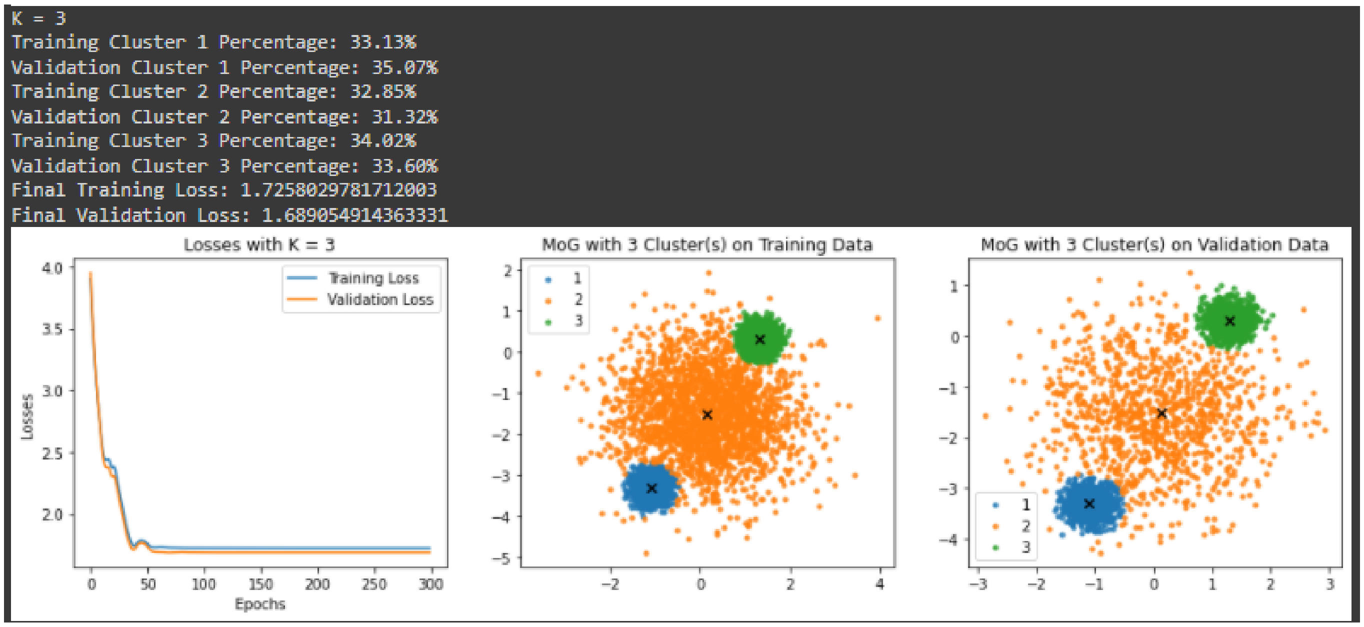
```
train_data, valid_data = load_2D(is_valid=True)

# train MoG model for 1,2,3,4,5
for K in range(1,6):
  print("K = {}".format(K))

  # compute losses
  train_loss, valid_loss, MU, sigma, log_pi= GMM(K, train_data, valid_data, is_valid=True

  # get clusters
  train_cluster = get_GMM_cluster(K, train_data, MU, log_pi, sigma)
  valid_cluster = get_GMM_cluster(K, valid_data, MU, log_pi, sigma)

  # get percentage of each cluster
  train_percentages = get_percentage(K, train_cluster)
  valid_percentages = get_percentage(K, valid_cluster)

  # plotting
  fig, ax = plt.subplots(1, 3, figsize=(16,4))

  # plotting loss
  ax[0].set_xlabel('Epochs')
  ax[0].set_ylabel('Losses')
  ax[0].plot(train_loss, label = "Training Loss")
  ax[0].plot(valid_loss, label = "Validation Loss")
  ax[0].set_title('Losses with K = {}'.format(K))
  ax[0].legend()

  # plotting scatters
  for i in range(K):
    train_x, train_y = train_cluster[i][:, 0], train_cluster[i][:, 1]
    valid_x, valid_y = valid_cluster[i][:, 0], valid_cluster[i][:, 1]

    # print percentage of each cluster
    print("Training Cluster {} Percentage: {:.2%}".format(i + 1,train_percentages[i]))
    print("Validation Cluster {} Percentage: {:.2%}".format(i + 1,valid_percentages[i]))

    # plot training scatter
    ax[1].scatter(train_x, train_y, s=10, alpha=0.8, label=str(i+1) )
    ax[1].plot(MU[:, 0], MU[:, 1], 'x', color='black')
    ax[1].set_title('MoG with {} Cluster(s) on Training Data'.format(K))
    ax[1].legend()

    # plot validation scatter
    ax[2].scatter(valid_x, valid_y, s=10, alpha=0.8, label=str(i+1) )
    ax[2].plot(MU[:, 0], MU[:, 1], 'x', color='black')
    ax[2].set_title('MoG with {} Cluster(s) on Validation Data'.format(K))
    ax[2].legend()

  print("Final Training Loss: {}".format(train_loss[-1]))
  print("Final Validation Loss: {}".format(valid_loss[-1]))

  plt.show()
```

```
K = 1
Training Cluster 1 Percentage: 100.00%
Validation Cluster 1 Percentage: 100.00%
Final Training Loss: 3.4897272960724406
Final Validation Loss: 3.4958965965425266
```



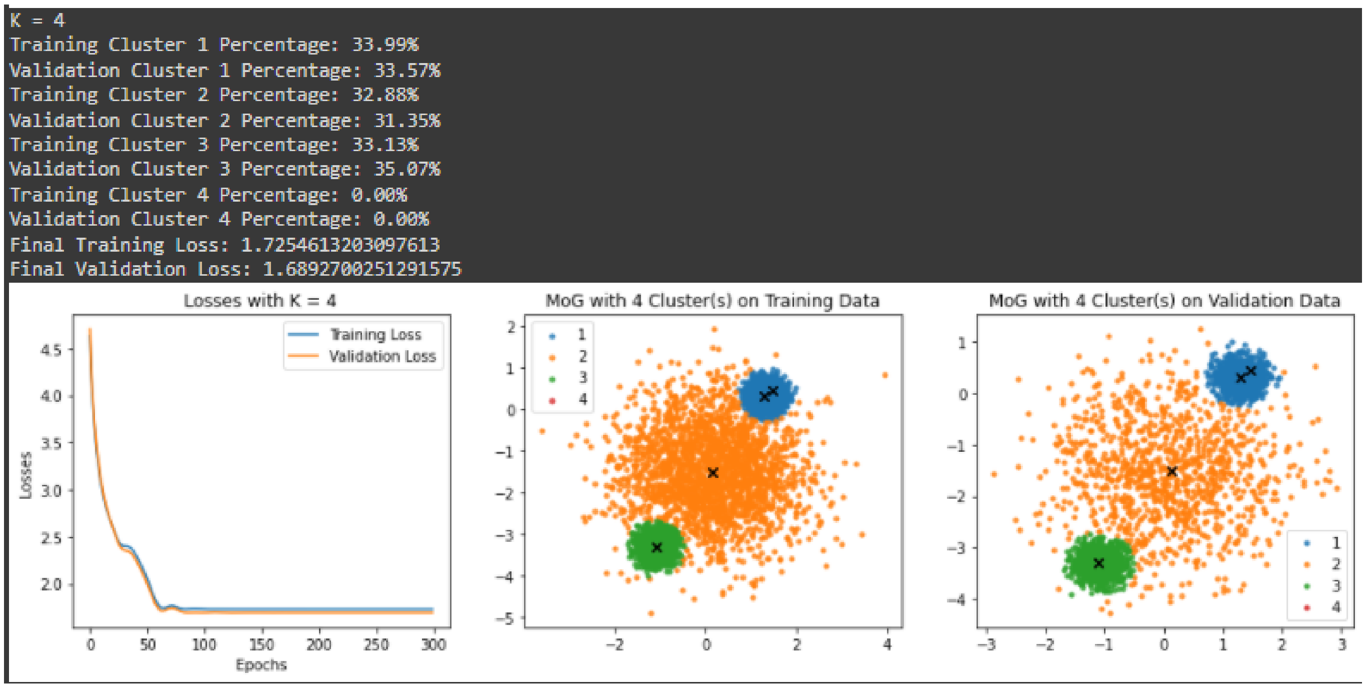
```
K = 2
Training Cluster 1 Percentage: 66.40%
Validation Cluster 1 Percentage: 64.30%
Training Cluster 2 Percentage: 33.60%
Validation Cluster 2 Percentage: 35.70%
Final Training Loss: 2.4387798408841848
Final Validation Loss: 2.404398879191213
```





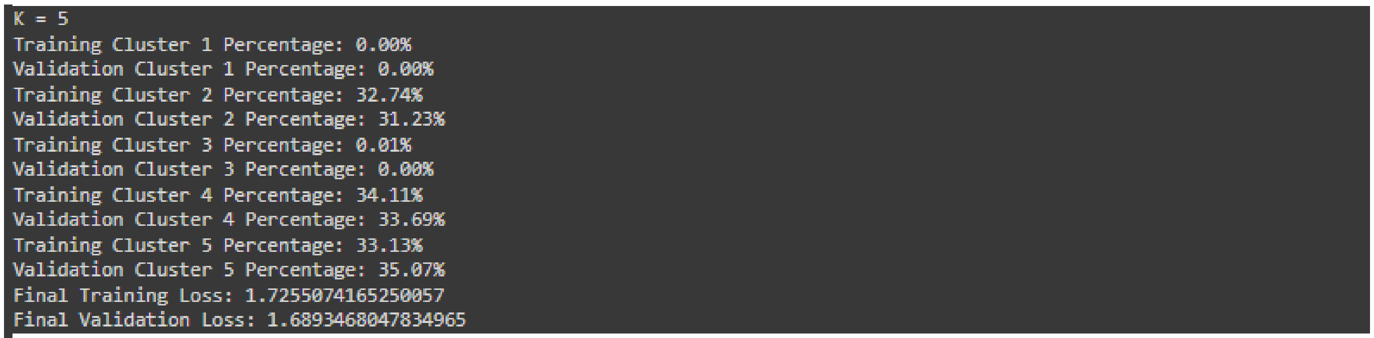
```
K = 3
Training Cluster 1 Percentage: 33.13%
Validation Cluster 1 Percentage: 35.07%
Training Cluster 2 Percentage: 32.85%
Validation Cluster 2 Percentage: 31.32%
Training Cluster 3 Percentage: 34.02%
Validation Cluster 3 Percentage: 33.60%
Final Training Loss: 1.7258029781712003
Final Validation Loss: 1.689054914363331
```

Losses with K = 3
Training Loss
Validation Loss
Losses
Epochs
MoG with 3 Cluster(s) on Training Data
MoG with 3 Cluster(s) on Validation Data
1
2
3

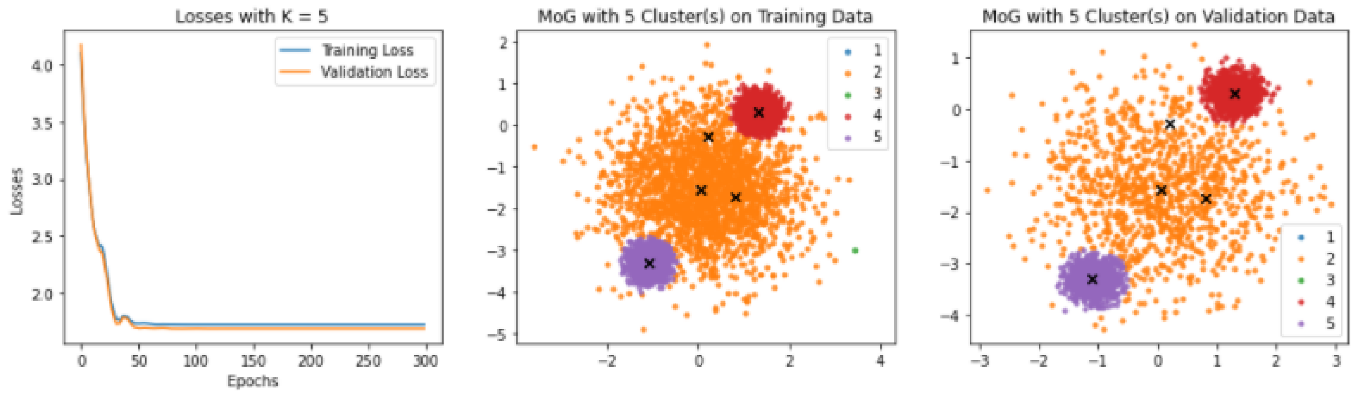


```
K = 4
Training Cluster 1 Percentage: 33.99%
Validation Cluster 1 Percentage: 33.57%
Training Cluster 2 Percentage: 32.88%
Validation Cluster 2 Percentage: 31.35%
Training Cluster 3 Percentage: 33.13%
Validation Cluster 3 Percentage: 35.07%
Training Cluster 4 Percentage: 0.00%
Validation Cluster 4 Percentage: 0.00%
Final Training Loss: 1.7254613203097613
Final Validation Loss: 1.6892700251291575
```

Losses with K = 4
Training Loss
Validation Loss
Losses
Epochs
MoG with 4 Cluster(s) on Training Data
MoG with 4 Cluster(s) on Validation Data
1
2
3
4



```
K = 5
Training Cluster 1 Percentage: 0.00%
Validation Cluster 1 Percentage: 0.00%
Training Cluster 2 Percentage: 32.74%
Validation Cluster 2 Percentage: 31.23%
Training Cluster 3 Percentage: 0.01%
Validation Cluster 3 Percentage: 0.00%
Training Cluster 4 Percentage: 34.11%
Validation Cluster 4 Percentage: 33.69%
Training Cluster 5 Percentage: 33.13%
Validation Cluster 5 Percentage: 35.07%
Final Training Loss: 1.7255074165250057
Final Validation Loss: 1.6893468047834965
```



The resulting percentages and final losses for each K is demonstrated here:

K	1	2	3	4	5
Final Training Loss	3.4897272960724406	2.4387798408841848	1.7258029781712003	1.7254613203097613	1.7255074165250057
Final Validation Loss	3.4958965965425266	2.404398879191213	1.689054914363331	1.6892700251291575	1.6893468047834965
Training Cluster Percentage	Cluster 1: 100 %	Cluster 1: 66.40% Cluster 2: 33.60%	Cluster 1: 33.13% Cluster 2: 32.85% Cluster 3: 34.02%	Cluster 1: 33.99% Cluster 2: 32.88% Cluster 3: 33.13% Cluster 4: 0.00%	Cluster 1: 0.00% Cluster 2: 32.74% Cluster 3: 0.01% Cluster 4: 34.11% Cluster 5: 33.13%
Validation Cluster Percentage	Cluster 1: 100%	Cluster 1: 64.30% Cluster 2: 35.70%	Cluster 1: 35.07% Cluster 2: 31.32% Cluster 3: 33.60%	Cluster 1: 33.57% Cluster 2: 32.88% Cluster 3: 33.13% Cluster 4: 0%	Cluster 1: 0.00% Cluster 2: 31.23% Cluster 3: 0.00% Cluster 4: 33.69% Cluster 5: 35.07%

Discussion

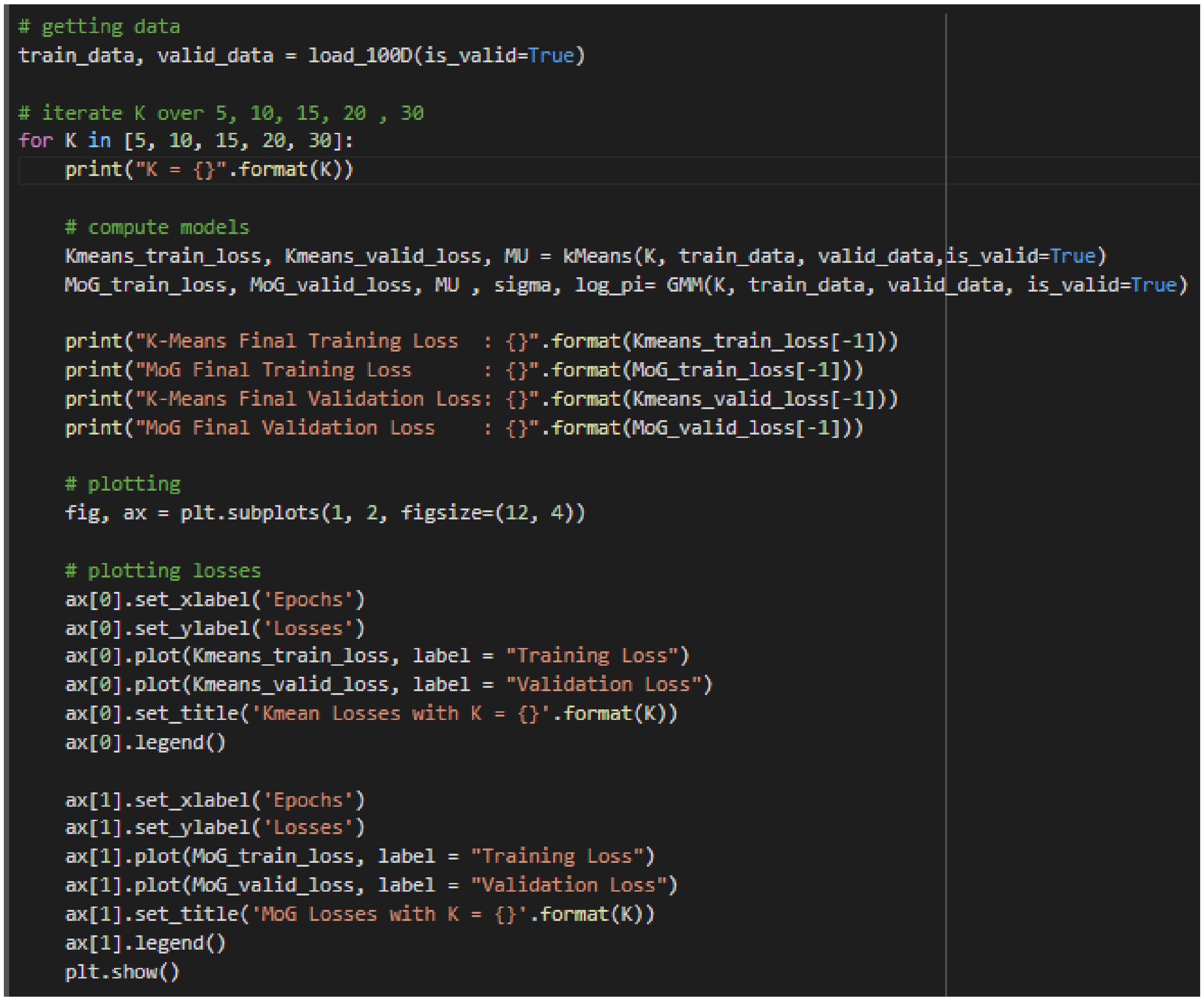
The best number of clusters to use is **3**.

When K =2, MoG model accurately determines one of the dense areas of the dataset. Despite the location of the dense cluster center, some data closer to this center actually belongs to the other cluster. This is how MoG is different from K-means.

When K =3, MoG model has 3 easily identifiable clusters with 2 on the corners and 1 scattered in the middle. However, for K > 3, the new clusters (specifically #4 and #5) are just added to the same location. The clusters beyond K =3 would be redundant clusters

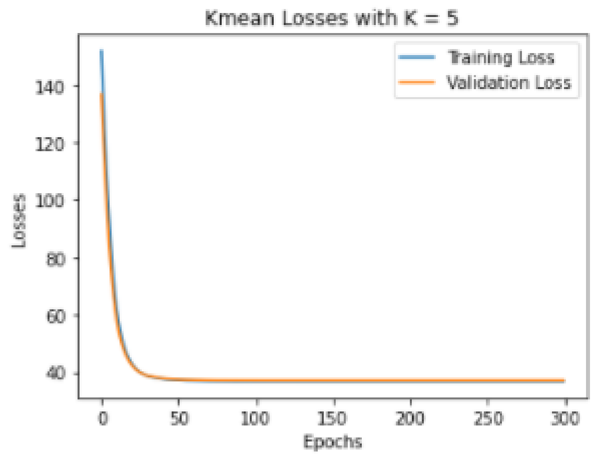
2.2.3 K-means vs. MoG

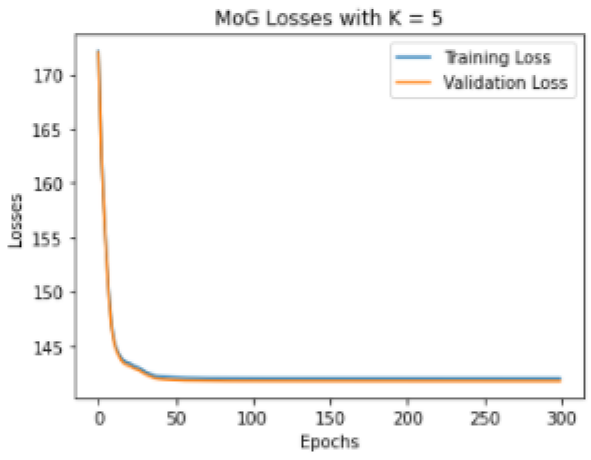
Run both the K-means and the MoG learning algorithms on data100D.npy for K = 5, 10, 15, 20, 30 while holding out ⅓ of the data for validation. Find the losses for both models.



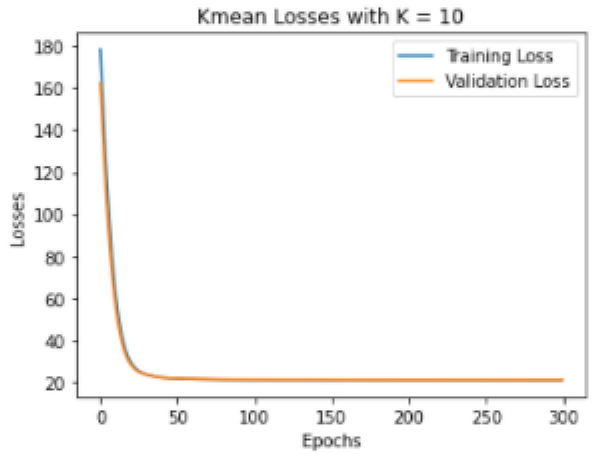
```
# getting data
train_data, valid_data = load_100D(is_valid=True)

# iterate K over 5, 10, 15, 20 , 30
for K in [5, 10, 15, 20, 30]:
    print("K = {}".format(K))

    # compute models
    Kmeans_train_loss, Kmeans_valid_loss, MU = kMeans(K, train_data, valid_data,is_valid=True)
    MoG_train_loss, MoG_valid_loss, MU , sigma, log_pi= GMM(K, train_data, valid_data, is_valid=True)

    print("K-Means Final Training Loss  : {}".format(Kmeans_train_loss[-1]))
    print("MoG Final Training Loss      : {}".format(MoG_train_loss[-1]))
    print("K-Means Final Validation Loss: {}".format(Kmeans_valid_loss[-1]))
    print("MoG Final Validation Loss    : {}".format(MoG_valid_loss[-1]))

    # plotting
    fig, ax = plt.subplots(1, 2, figsize=(12, 4))

    # plotting losses
    ax[0].set_xlabel('Epochs')
    ax[0].set_ylabel('Losses')
    ax[0].plot(Kmeans_train_loss, label = "Training Loss")
    ax[0].plot(Kmeans_valid_loss, label = "Validation Loss")
    ax[0].set_title('Kmean Losses with K = {}'.format(K))
    ax[0].legend()

    ax[1].set_xlabel('Epochs')
    ax[1].set_ylabel('Losses')
    ax[1].plot(MoG_train_loss, label = "Training Loss")
    ax[1].plot(MoG_valid_loss, label = "Validation Loss")
    ax[1].set_title('MoG Losses with K = {}'.format(K))
    ax[1].legend()
    plt.show()
```

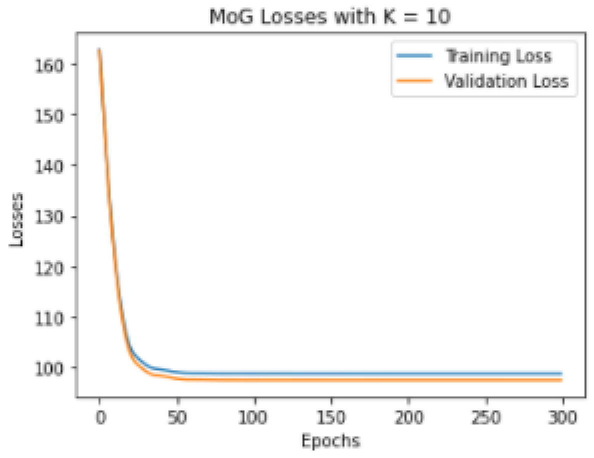
```
K = 5
K-Means Final Training Loss  : 36.86315225760681
MoG Final Training Loss      : 141.98310952968998
K-Means Final Validation Loss: 37.123499839353535
MoG Final Validation Loss    : 141.73298491965937
```



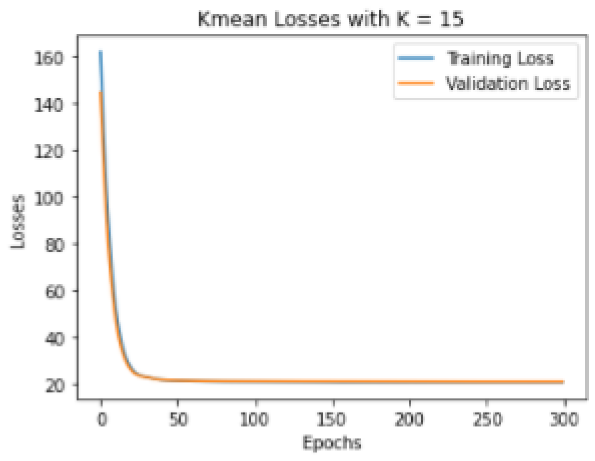


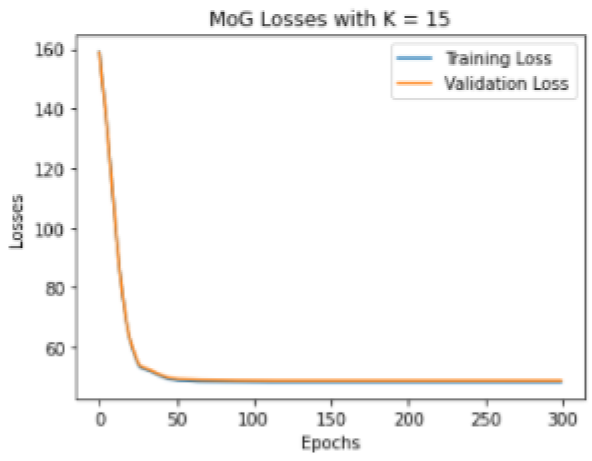
```
K = 10
K-Means Final Training Loss  : 21.045965130952073
MoG Final Training Loss      : 98.74017935873927
K-Means Final Validation Loss: 21.139840907145516
MoG Final Validation Loss    : 97.43823463508645
```



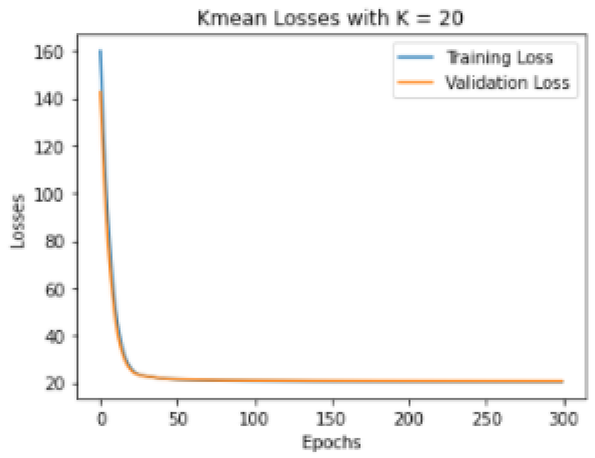


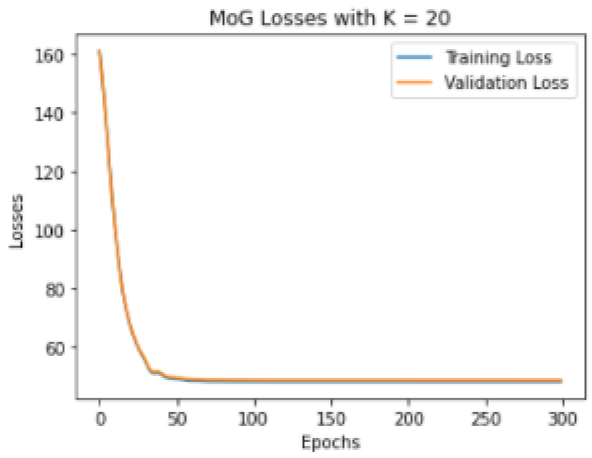
```
K = 15
K-Means Final Training Loss  : 20.748376591265167
MoG Final Training Loss      : 48.28946246076851
K-Means Final Validation Loss: 20.89194825464193
MoG Final Validation Loss    : 48.77025336963775
```



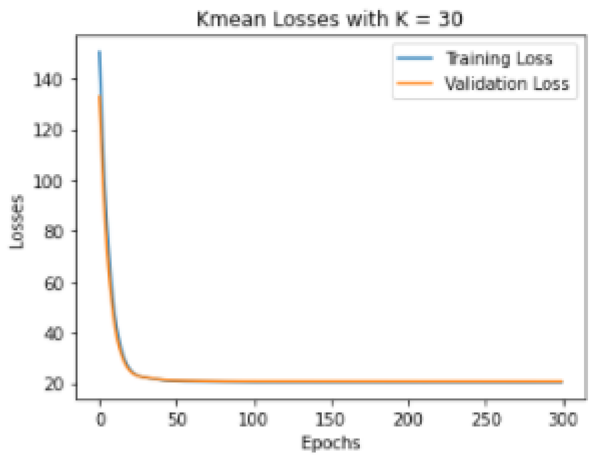


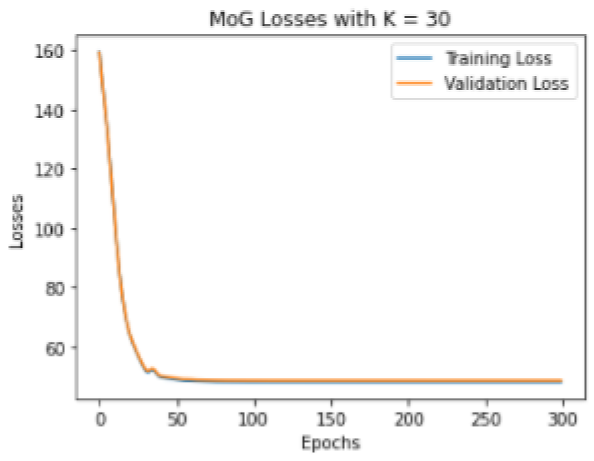
```
K = 20
K-Means Final Training Loss  : 20.487704363197253
MoG Final Training Loss      : 48.28468805583286
K-Means Final Validation Loss: 20.68583721019146
MoG Final Validation Loss    : 48.78960634247146
```





```
K = 30
K-Means Final Training Loss  : 20.45245788066806
MoG Final Training Loss      : 48.28341572680684
K-Means Final Validation Loss: 20.71041236054479
MoG Final Validation Loss    : 48.783699410876885
```





Discussion

I think there are **15** clusters within the 100D dataset.

For the K-means models, a larger K would allow better partition of the input data. The average distance from each data point to its cluster center is decreased since there are more clusters.

The MoG models can be more helpful in determining the number of clusters. Unlike the K-means models in which data are split evenly by a hyperplane bisecting the 2 nearest clusters, MoG models assign data to the cluster with the higher probability. From the 2D dataset, we observed that K increases beyond the desired number of clusters. Extra clusters are just added to the same location repeatedly after the desired number is reached. Thus, increasing K beyond optimal cluster number would not result in better loss values for MoG models. As we can see in the output plots, when K=5 or 10, we do not have enough clusters. For K= 15, 20, or 30, the losses all converge around 48. This means increasing K beyond 15 does not have a significant decrease in loss which infers the extra clusters are all redundant.

MoG models appear to have worse losses for smaller K since K-means can partition unidentifiable clusters better. However, even though K-means models have lower loss values from MoG models, K-Means can have its drawbacks. For example, if K = # of data points in the dataset, K-means would achieve a loss = 0 which means it assigns each data point to its own cluster. This is overfitting and should be prevented. On the other hand, MoG models would give similar results for K > 15 and beyond. In conclusion, K-Means models are better for smaller K and MoG models are better for larger K.