In [1]:
```
import operator
import sys, os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import linalg
from sklearn.manifold import TSNE
from pyspark import SparkConf, SparkContext
```

```
/opt/conda/anaconda/lib/python3.7/site-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. U
se the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

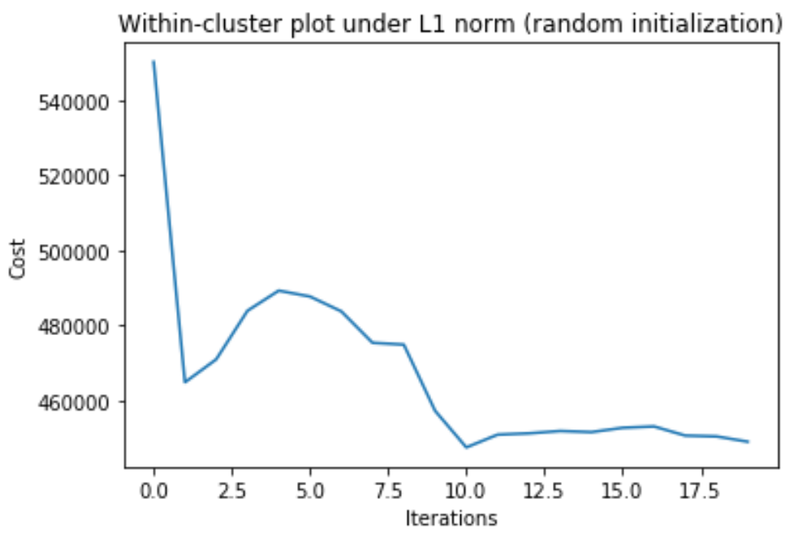
In [2]:
```
# Macros.
MAX_ITER = 20
DATA_PATH = "gs://6893_course_data/hw1_q1/data.txt"
C1_PATH = "gs://6893_course_data/hw1_q1/c1.txt"
C2_PATH = "gs://6893_course_data/hw1_q1/c2.txt"
L1 = 1
L2 = 2
```

In [10]:
```
# Helper functions
def closest(p, centroids, norm):
    """
    Compute closest centroid for a given point.
    Args:
        p (numpy.ndarray): input point
        centroids (list): A list of centroids points
        norm (int): 1 or 2
    Returns:
        int: The index of closest centroid.
    """
    closest_c = min([(i, linalg.norm(p - c, norm))
                     for i, c in enumerate(centroids)],
                    key=operator.itemgetter(1))[0]
    return closest_c


def within_cluster_cost(data, centroids, norm):
    """
    Compute within-cluster cost.
    Args:
        data (RDD): a RDD of the form (centroid, (point, 1))
        centroids (list): A list of centroids points
        norm (int): 1 or 2
    Returns:
        Float: Within-cluster cost of the current classification.
    """
    cost = sum(data.map(lambda pt: linalg.norm(pt[1][0] - centroids[pt[0]], norm)).collect())
    return cost


def tSNE_vis(data, norm, init):
    """
    Produce a t-SNE 2D dimensional clustering result.
    Args:
        data (RDD): a RDD of (centroid, (point, 1))
        norm (int): 1 or 2
        init (str): 'random' or 'furthest'
    Returns:
        None
    """
    arr = np.array(data.map(lambda pt: list(pt[1][0])).collect())
    labels = data.keys().collect()
    embedded = TSNE(n_components=2, random_state=666).fit_transform(arr)
    x = embedded[:, 0]
    y = embedded[:, 1]
    plt.scatter(x, y, c=labels)
    plt.title('t-SNE plot under L%d norm (%s initialization)'%(norm, init))
    plt.show()
```

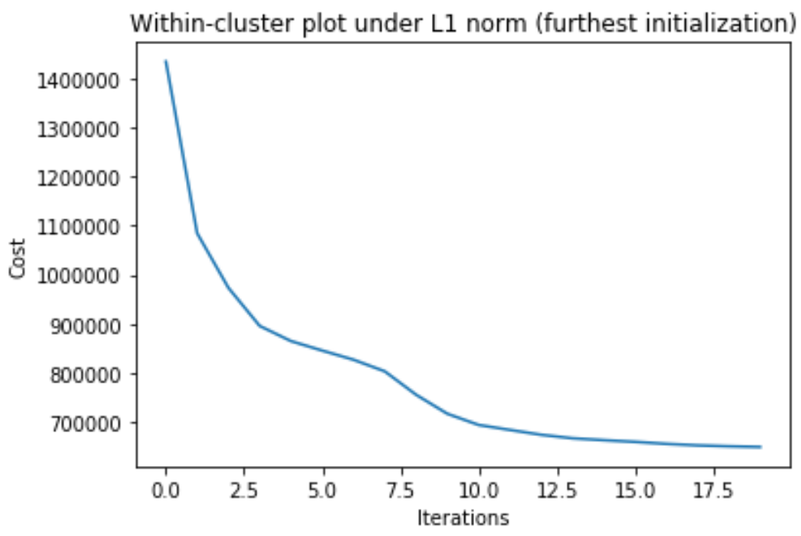
In [4]:
```python
# K-means clustering
def kmeans(data, centroids, norm, init):
    """
    Conduct k-means clustering given data and centroid.
    Args:
        data (RDD): RDD of points
        centroids (list): A list of centroids points
        norm (int): 1 or 2
        init (str): 'random' or 'furthest'
    Returns:
        RDD: assignment information of points, a RDD of (centroid, (point, 1))
        list: a list of centroids
    """

    cost_list = np.zeros(MAX_ITER)

    # iterative k-means
    for i in range(MAX_ITER):

        # Points assignment
        points = data.map(lambda p: (closest(p, centroids, norm), (p, 1)))

        # Cost Calculation
        cost_list[i] = within_cluster_cost(points, centroids, norm)

        # Updata centroids
        reduced_pts = points.reduceByKey(lambda p1, p2: (p1[0] + p2[0], p1[1] + p2[1]))
        centroids = reduced_pts.values().map(lambda c: c[0] / c[1]).collect()

    # cost plot
    plt.plot(cost_list)
    plt.xlabel('Iterations')
    plt.ylabel('Cost')
    plt.title('Within-cluster plot under L%d norm (%s initialization)'%(norm, init))
    plt.show()

    return points, centroids
```

In [5]:
```python
# Spark settings
/gateway/default/node/conf?host&port = SparkConf()
sc = SparkContext(/gateway/default/node/conf?host&port=/gateway/default/node/conf?host&port)

# Load the data, cache this since we're accessing this each iteration
data = sc.textFile(DATA_PATH).map(
        lambda line: np.array([float(x) for x in line.split(' ')])
        ).cache()
# Load the initial centroids c1, split into a list of np arrays
centroids1 = sc.textFile(C1_PATH).map(
        lambda line: np.array([float(x) for x in line.split(' ')])
        ).collect()
# Load the initial centroids c2, split into a list of np arrays
centroids2 = sc.textFile(C2_PATH).map(
        lambda line: np.array([float(x) for x in line.split(' ')])
        ).collect()
```

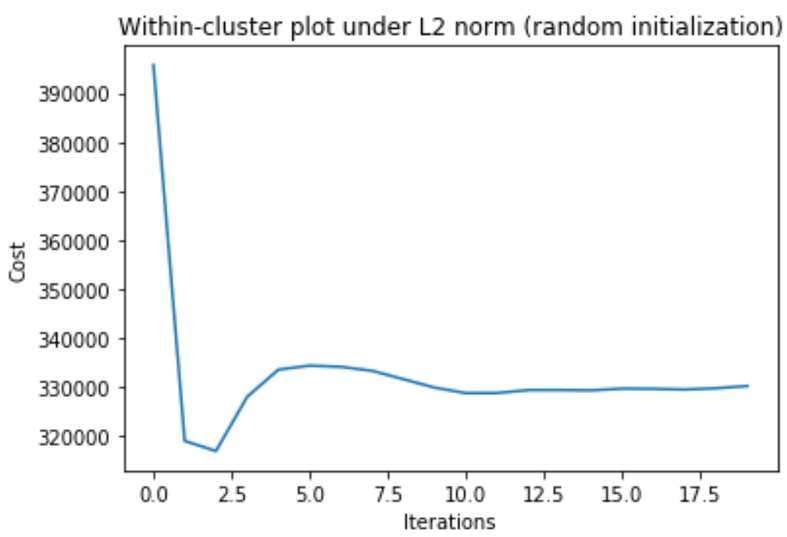
In [11]:
```python
# run kmeans
clustered_1, updated_centroids_1 = kmeans(data, centroids1, L1, 'random')
```

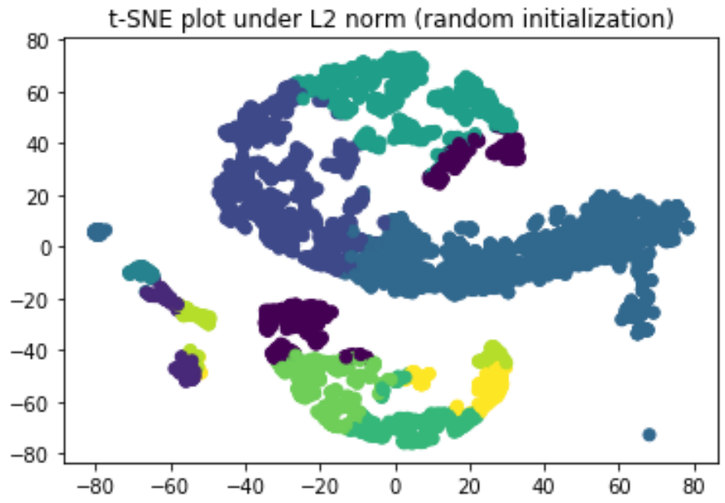


In [12]:
```python
# run kmeans
clustered_2, updated_centroids_2 = kmeans(data, centroids2, L1, 'furthest')
```



In [13]:
```
# run kmeans
clustered_3, updated_centroids_3 = kmeans(data, centroids1, L2, 'random')
# t-SNE visualization
tSNE_vis(clustered_3, L2, 'random')
```





In [14]:
```
# run kmeans
clustered_4, updated_centroids_4 = kmeans(data, centroids2, L2, 'furthest')
# t-SNE visualization
tSNE_vis(clustered_4, L2, 'furthest')
```