E-tivity 3: Clustering and Manifold Learning

Name: Martin Power

ID : 9939245 ¶

Use this notebook to complete Tasks 1 and 2 in E-tivity3.

Import Python Modules

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn import manifold
from sklearn import cluster
from sklearn import preprocessing
from sklearn.preprocessing import power_transform
from sklearn.metrics import silhouette_samples
from sklearn.metrics import silhouette_score
from matplotlib.ticker import FixedLocator
from matplotlib.ticker import FixedFormatter
```

Table of Contents

Task 1 (CS5062)

Inspect and Analyse Dataset to begin with

In [2]:

```
df = pd.read_csv("./loans_dataset_et3.csv")

print("Number of Samples  in Dataset:\t",df.shape[0])
print("Number of Features in Dataset:\t",df.shape[1])
```

```
Number of Samples  in Dataset:	 332
Number of Features in Dataset:	 5
```

In [3]:

```
df.head()
```

Out[3]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	2483	2466.0	90	180	0
1	4917	0.0	130	360	0
2	4106	0.0	40	180	1
3	3859	3300.0	142	180	1
4	6417	0.0	157	180	1

In [4]:

```
df.tail()
```

Out[4]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
327	5417	4196.0	267	360	1
328	16666	0.0	275	360	1
329	10750	0.0	312	360	1
330	5955	5625.0	315	360	1
331	6133	3906.0	324	360	1

In [5]:

```
# Print statistical summary for all attributes
df.describe(include='all')
```

Out[5]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_Histo
count	332.000000	332.000000	332.000000	332.000000	332.0000
mean	5201.093373	1495.508795	140.882530	341.710843	0.9789
std	4584.815491	1982.742932	75.544237	61.651497	0.1438
min	645.000000	0.000000	17.000000	60.000000	0.0000
25%	2912.750000	0.000000	100.000000	360.000000	1.0000
50%	3858.500000	1211.500000	128.000000	360.000000	1.0000
75%	5818.250000	2250.000000	162.000000	360.000000	1.0000
max	39999.000000	20000.000000	600.000000	480.000000	1.0000

In [6]:

```
# Quick Check to Ensure no missing data
print(df.isnull().any())
```

```
ApplicantIncome      False
CoapplicantIncome    False
LoanAmount           False
Loan_Amount_Term     False
Credit_History       False
dtype: bool
```

Visualise Distribution of Features

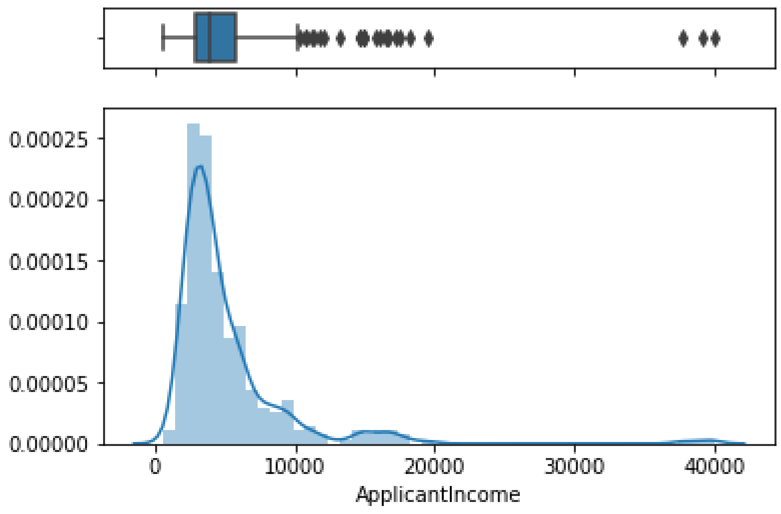
In [7]:

```
# Function from previous Etivity to plot histogram and boxplot
def plot_hist_with_box(feature):
    # From https://python-graph-gallery.com/24-histogram-with-a-boxplot-on-top-seaborn/
    # Cut the window in 2 parts
    f, (ax_box, ax_hist) = plt.subplots(2, sharex=True, gridspec_kw={"height_ratios": (
.15, .85)})

    # Add a graph in each part
    sns.boxplot(feature, ax=ax_box)
    sns.distplot(feature, ax=ax_hist)

    # Remove x axis name for the boxplot
    ax_box.set(xlabel='')
    plt.show()
    return
```

In [8]:

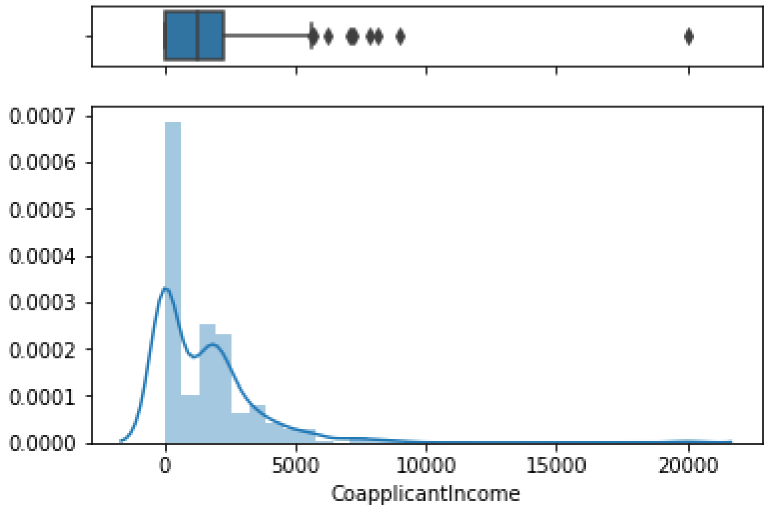
```
# Plot ApplicantIncome
plot_hist_with_box(df['ApplicantIncome'])
```

```
C:\Users\mpower1\AppData\Local\Continuum\anaconda3\lib\site-packages\scipy
\stats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multid
imensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[s
eq]`. In the future this will be interpreted as an array index, `arr[np.ar
ray(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



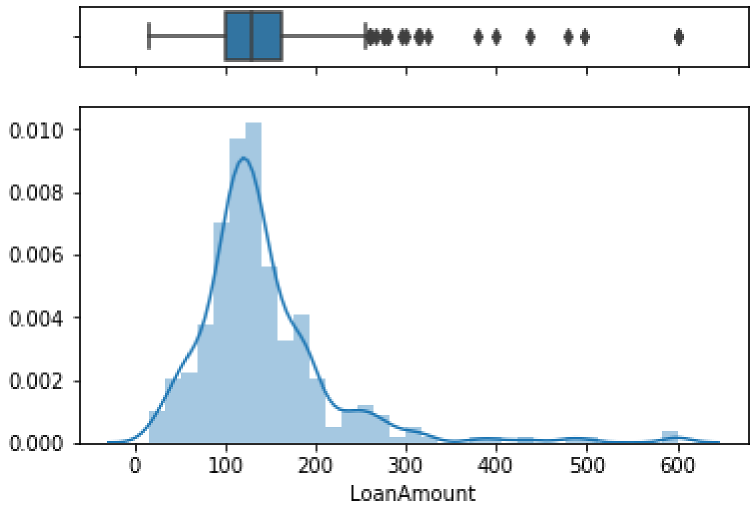
In [9]:

```
# Plot CoapplicantIncome
plot_hist_with_box(df['CoapplicantIncome'])
```



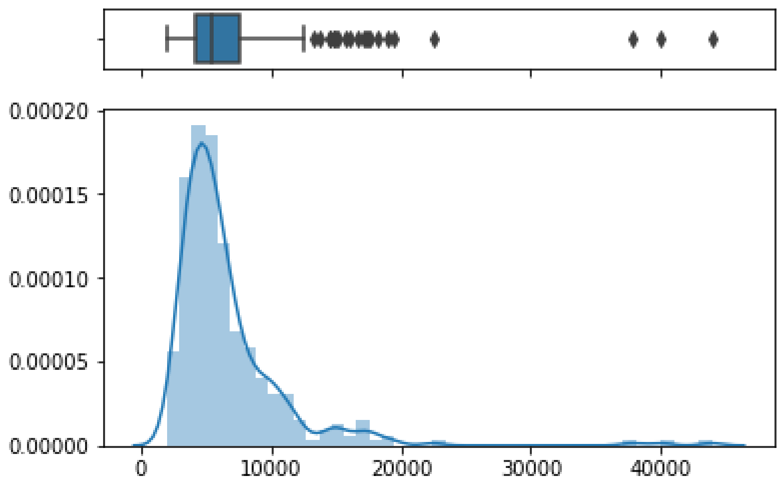
In [10]:

```
# Plot LoanAmount
plot_hist_with_box(df['LoanAmount'])
```



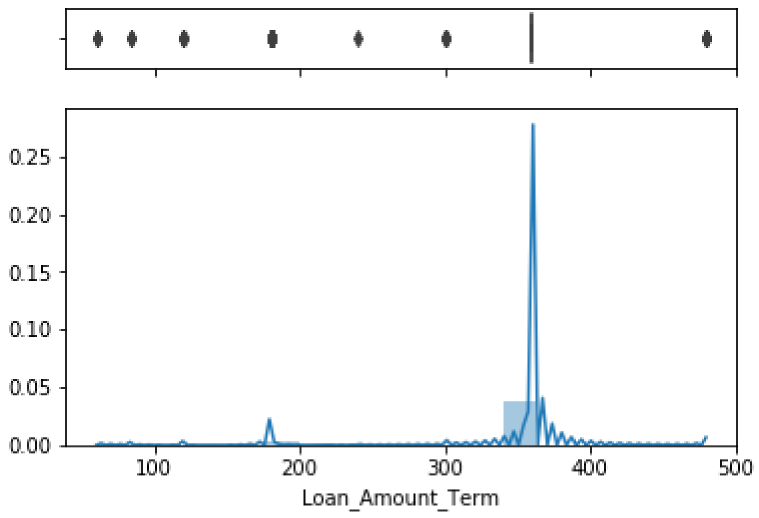
In [11]:

```
# Plot Applicant Income and Coapplicant Income
# Looking at this for potential "JointIncome" Feature
plot_hist_with_box(df['ApplicantIncome']+df['CoapplicantIncome'])
```



In [12]:

```
plot_hist_with_box(df['Loan_Amount_Term'])
print("Number of Unique Loan Amount Term Values is ",df['Loan_Amount_Term'].nunique())
print("Unique Values and Counts:")
df['Loan_Amount_Term'].value_counts()
```



```
Number of Unique Loan Amount Term Values is  8
Unique Values and Counts:
```

Out[12]:

```
360    292
180     24
480      4
300      4
120      3
84       2
60       2
240      1
Name: Loan_Amount_Term, dtype: int64
```

In [13]:

```
# We know that Credit History only has 2 values, 0 and 1, so plotting distribution is n
ot needed
# Instead, see what the count of 0s and 1s looks like

print("Number of Unique Credit History values are ",df['Credit_History'].nunique())
print("Unique Values and Counts:")
df['Credit_History'].value_counts()
```

```
Number of Unique Credit History values are  2
Unique Values and Counts:
```

Out[13]:

```
1    325
0      7
Name: Credit_History, dtype: int64
```

Summary of Distribution of Features

- Applicant Income
  - Normal-like Distribution with a large cluster of near outliers and some far outliers
- Coapplicant Income
  - Large number of 0s in the data for all examples that only had a single applicant
  - This skews the distribution
  - Distribution looks much better for a "JointIncome" feature
- LoanAmount
  - Normal distribution
  - Many outliers at upper end of distribution
- Loan_Amount_Term
  - Does not have a normal or uniform distribution
  - 8 unique values, all multiples of 12 (i.e terms are in years)
  - 88% of values are 360 months
  - 7% of values are 180 months
  - Remaining 5% of values are distributed across 6 values
  - Feature is quite skewed to two values and this may not suit k-means algorithm
- Credit History
  - Only 2 values - 0 and 1
  - 97.9% of dataset has a value of 1 for Credit History
  - Only 7 out of 332 examples have a value of 0 from Credit History
  - Feature is not at all balanced within dataset and this may not suit k-means algorithm

Task 1 - Feature Scaling

- I created a post on the Etivity3 forum entitled "Task 1 and Task 2 : Observations about Scaling and Possible Feature Loss" where I discuss my experience trying to apply transforms to the features to account for outliers and distribution - see https://sulis.ul.ie/portal/directtool/e62cef69-95cc-4727-945a-40935a1c2038/ (https://sulis.ul.ie/portal/directtool/e62cef69-95cc-4727-945a-40935a1c2038/)

- I found that my clustering results degraded when I applied transforms and that features were "lost". Specifically, Loan_Amount_Term and Credit_History were influencing the clustering once the transforms were applied

- I have conflicting thoughts on this. On the one hand, these features are quite imbalanced within the dataset and you could argue that it is better for the clustering to almost ignore these features as they are so imbalanced. On the other hand, it may be still meaningful to try and cluster based on these features.
- In the end, I went with simple MinMax scaling as this kept all features within a common range and appeared to produce the most interesting clusters

In [14]:

```
# Reusing code from Lab3

# Fixed typo in code where 'blue' appeared twice. Replaced with 'yellow'
colors = np.array(['orange', 'blue', 'lime', 'yellow', 'khaki', 'pink', 'green', 'purpl
e'])

# points - a 2D array of (x,y) coordinates of data points
# labels - an array of numeric labels in the interval [0..k-1], one for each point
# centers - a 2D array of (x, y) coordinates of cluster centers
# title - title of the plot

def clustering_scatterplot(points, labels, centers, title):
    # plot the examples, i.e. the data points

    n_clusters = np.unique(labels).size
    for i in range(n_clusters):
        h = plt.scatter(points[labels==i,0],
                        points[labels==i,1],
                        c=colors[i%colors.size],
                        label = 'cluster '+str(i))

    # plot the centers of the clusters
    if centers is not None:
        plt.scatter(centers[:,0], centers[:,1], c='r', marker='*', s=500)

    _ = plt.title(title)
    _ = plt.legend()
    _ = plt.xlabel('x')
    _ = plt.ylabel('y')
```

In [15]:

```
# Create Matrix of data
scale_data = np.array(df.values, dtype=float)
print('(number of examples, number of attributes): ', scale_data.shape)
```

```
(number of examples, number of attributes):  (332, 5)
```

Apply Min-Max Scaling

In [16]:

```
min_max_scaler = preprocessing.MinMaxScaler()
scale_data = min_max_scaler.fit_transform(scale_data)
```

In [17]:

```
# Do a quick visual check on the first five rows of MinMax scale_data and original cont
ents
print(scale_data[:5])
df.head()
```

```
[[0.04670427 0.1233     0.12521441 0.28571429 0.        ]
 [0.10855313 0.         0.19382504 0.71428571 0.        ]
 [0.08794532 0.         0.03945111 0.28571429 1.        ]
 [0.08166895 0.165      0.21440823 0.28571429 1.        ]
 [0.1466687  0.         0.24013722 0.28571429 1.        ]]
```

Out[17]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	2483	2466.0	90	180	0
1	4917	0.0	130	360	0
2	4106	0.0	40	180	1
3	3859	3300.0	142	180	1
4	6417	0.0	157	180	1

Run K-Means on Scaled Data

In [18]:

```
# K-Means Parameters
# Choosing K=4 for Initial Run
k = 4

# Number of time the k-means algorithm will be run with different centroid seeds.
# The final results will be the best output of n_init consecutive runs in terms of iner
tia.
n_init = 20

# Maximum number of iterations of the k-means algorithm for a single run.
max_iter = 500
random_state = 0 # Use this to make results repeable for analysis in Markdown cells
```

In [19]:

```
clustered_data_sklearn = cluster.KMeans(n_clusters=k, n_init=n_init, max_iter=max_iter,
random_state=random_state).fit(scale_data)
```

In [20]:

```
# append the cluster centers to the dataset
scale_data_and_centers = np.r_[scale_data,clustered_data_sklearn.cluster_centers_]
```

Task 1 - MDS Visualisation

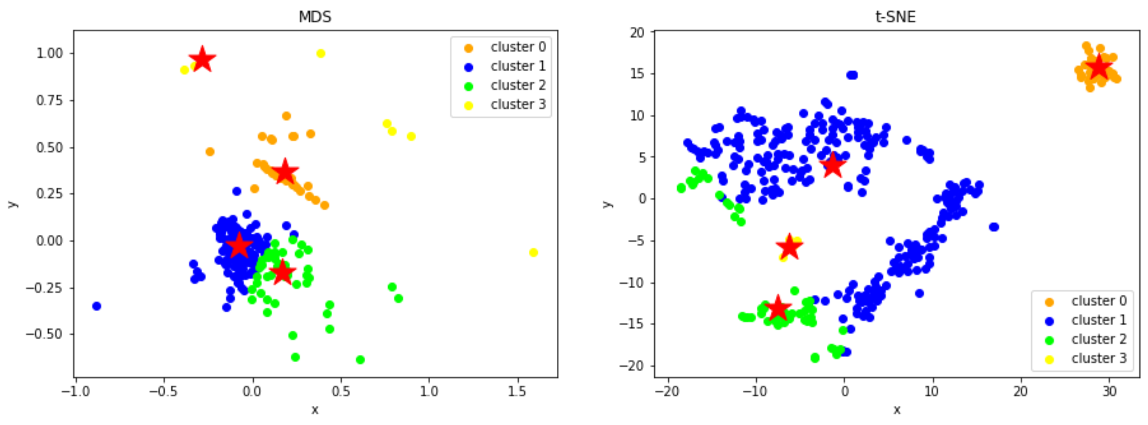
Task 1 - t-SNE Visualisation

In [21]:

```
def plot_mds_tsne(n_clusters, plot_data, plot_labels, n_components=2, random_state=0):
    # Plot MDS and t-SNE visualisations for data

    plt.subplots(1, 2, figsize=(15, 5))

    # MDS Plot
    plt.subplot(1,2,1)
    # project both th data and the k-Means cluster centers to a 2D space
    XYcoordinates = manifold.MDS(n_components=n_components, random_state=random_state).
fit_transform(plot_data)
    clustering_scatterplot(points=XYcoordinates[:-n_clusters,:],
                           labels=plot_labels,
                           centers=XYcoordinates[-n_clusters:,:],
                           title='MDS')

    # t-SNE Plot
    plt.subplot(1,2,2)
    XYcoordinates = manifold.TSNE(n_components=n_components, random_state=random_state)
.fit_transform(plot_data)
    clustering_scatterplot(points=XYcoordinates[:-n_clusters,:],
                           labels=plot_labels,
                           centers=XYcoordinates[-n_clusters:,:],
                           title='t-SNE')
    # Plot graphs
    plt.show()
```

In [22]:

```
plot_mds_tsne(k, scale_data_and_centers, clustered_data_sklearn.labels_, n_components=2
)
```



MDS and t-SNE Plot Analysis (*K*=4)

- Cluster 1 and Cluster 2 are very close in both plots
  - This suggests that elements might be similar
  - This will turn out to be true in later analysis
- Both plots have Cluster 3 (CreditHistory=0) and Cluster 0 (Smaller, Shorter Loans) as very distinct groupings

In [23]:

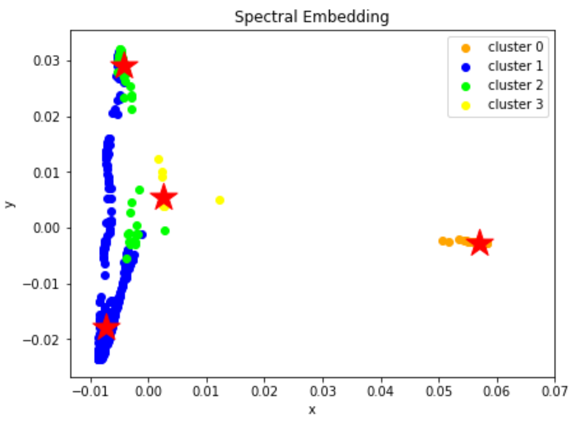
```
def plot_se_lle(n_clusters, n_neighbors,plot_data, plot_labels, n_components=2, random_
state=0):
    # Plot Spectral Embedding and  Locally Linear Embedding visualisations for data

    plt.subplots(1, 2, figsize=(15, 5))

    # SE Plot
    plt.subplot(1,2,1)
    XYcoordinates = manifold.SpectralEmbedding(n_components=n_components,n_neighbors=n_
neighbors,random_state=random_state).fit_transform(plot_data)
    clustering_scatterplot(points=XYcoordinates[:-n_clusters,:],
                           labels=plot_labels,
                           centers=XYcoordinates[-n_clusters:,:],
                           title='Spectral Embedding')

    # LLE Plot
    plt.subplot(1,2,2)
    XYcoordinates = manifold.LocallyLinearEmbedding(n_components=n_components,n_neighbo
rs=n_neighbors,random_state=random_state).fit_transform(plot_data)
    clustering_scatterplot(points=XYcoordinates[:-n_clusters,:],
                           labels=plot_labels,
                           centers=XYcoordinates[-n_clusters:,:],
                           title='Locally Linear Embedding')
    # Plot graphs
    plt.show()
```

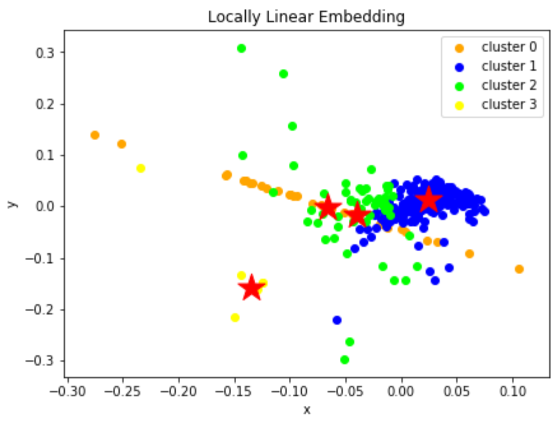
Task 2 - Additional Manifold Learning Technique

- I have decided to also experiment with using Spectral Embedding and Local Linear Embedding Manifold Learning Techniques

In [24]:

```
plot_se_lle(k, 10, scale_data_and_centers, clustered_data_sklearn.labels_, n_components
=2)
```





SE and LLE Plot Analysis

- Cluster 1 and Cluster 2 are very close in both plots, similar to MDS and t-SNE
- LLE has Cluster 0 passing through Cluster 1 and Cluster 2 - perhaps it seems Cluster 0 as subset
- LLE has Cluster 3 (CreditHistory=0) and very distinct group although SE does not make it as distinct

Task 1 - Cluster Description (k=4)

In [25]:

```
# Append the cluster labels to the original data
df['cluster_k4'] = pd.Series(clustered_data_sklearn.labels_, index=df.index)
```

In [26]:

```
# Print the number of members of each cluster
df['cluster_k4'].value_counts()
```

Out[26]:

```
1    241
2     54
0     30
3      7
Name: cluster_k4, dtype: int64
```

In [27]:

```
df.groupby('cluster_k4').mean()
```

Out[27]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_
cluster_k4					
0	5146.966667	1297.126666	116.333333	161.600000	
1	3829.493776	1541.436183	118.419087	361.493776	
2	10840.166667	1322.629630	246.296296	357.777778	
3	9153.857143	2098.142857	206.285714	308.571429	

Task 1 - Cluster Characteristics (k=4)

Cluster 0

- 30 Entries
- Joint Income is approx 6000
- LoanAmount is below the mean amount of 140 for the entire dataset
- LoanTerm is below the mean amount of 341 for the entire dataset
- Credit History for all entrants is 1
    - ***Low Risk, Average Income, Low Amount, Short Term Loan***

Cluster 1

- 241 Entries
- Joint Income is approx 5000
- LoanAmount is below the mean amount of 140 for the entire dataset
- LoanTerm is just above the mean amount of 341 for the entire dataset
- Credit History for all entrants is 1
    - ***Low Risk, Average Income, Average Amount, Long Term Loan***

Cluster 2

- 54 Entries
- Joint Income is approx 12000 and main ApplicantIncome is almost 11K - these are outliers in ApplicantIncome boxplot
- LoanAmount mean of 246 is well above the mean amount of 140 for entire dataset - these are outlier in LoanAmount boxplot
- LoanTerm is just above the mean amount of 341 for the entire dataset
- Credit History for all entrants is 1
    - ***Low Risk, High Income, High Amount, Long Term Loan***

Cluster 3

- 7 Entries
- Joint Income is approx 11000 and main ApplicantIncome is 9K
- LoanAmount mean of 206 is above the mean amount of 140 for entire dataset
- LoanTerm is below the mean amount of 341 for the entire dataset
- Credit History for all entrants is 1
    - ***High Risk, High Income, Medium Amount, Medium Term Loan***

Based on the above, the clusters can be characterised as follows:

- **Cluster 0:** Low Risk, Average Income, Low Amount, Short Term Loan
- **Cluster 1:** Low Risk, Average Income, Average Amount, Long Term Loan
- **Cluster 2:** Low Risk, High Income, High Amount, Long Term Loan
- **Cluster 3:** High Risk, High Income, Medium Amount, Medium Term Loan

Task 2 (CS5062)

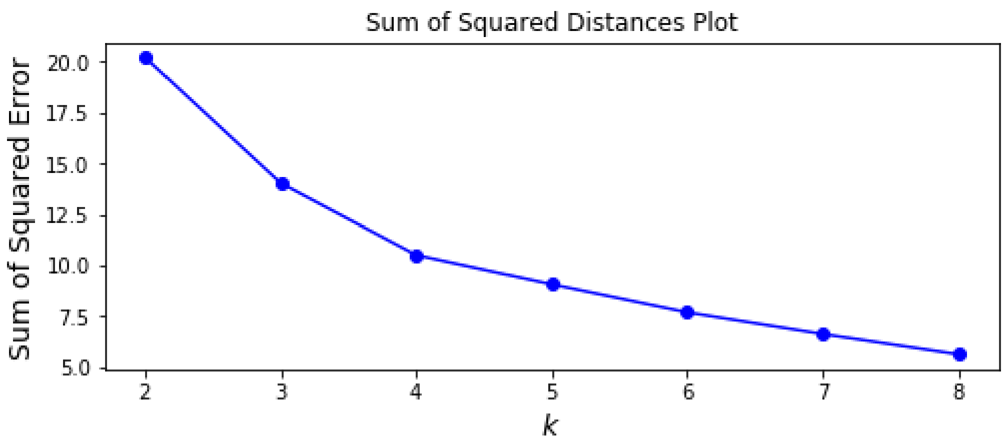
Task 2 - Sum of Squared Distances Plot

In [28]:

```
# Sweep from K=2 up to K=8
min_k = 2
max_k = 8

kvals = np.array(range(min_k,max_k+1))
```

In [29]:

```
sse = np.empty(len(kvals))

for i in range(len(kvals)):
    clustered_data_sklearn = cluster.KMeans(n_clusters=kvals[i],  n_init=n_init, max_it
er=max_iter, random_state=random_state).fit(scale_data)

    # Store Sum of Squared Error Value
    sse[i] = clustered_data_sklearn.inertia_


plt.figure(figsize=(8, 3))
plt.title("Sum of Squared Distances Plot")
plt.plot(kvals,  sse, "bo-")
plt.xlabel("$k$", fontsize=14)
plt.ylabel("Sum of Squared Error", fontsize=14)
plt.show()
```



Task 2 - Elbow Method to Find Best k

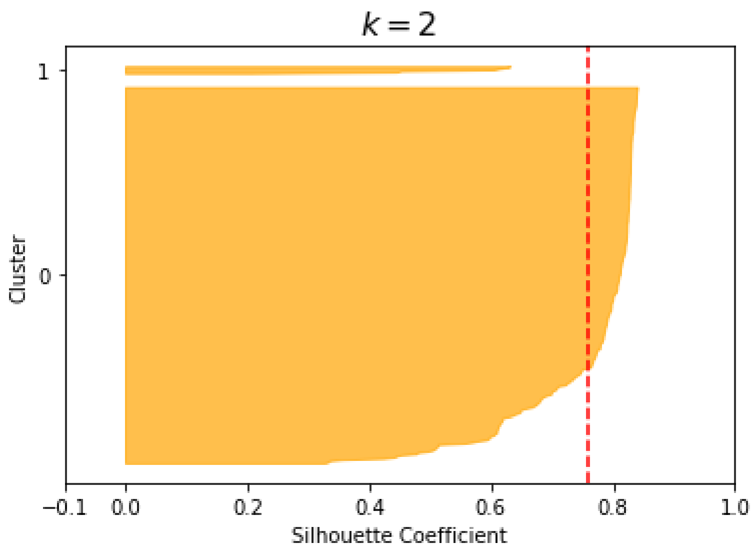
- See below "Task 2 - Silhouette Coefficient Method to Find Best k"

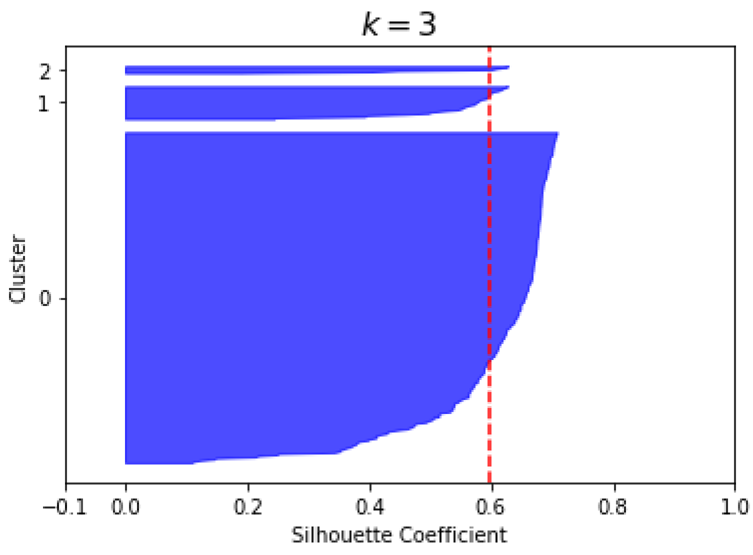
Task 2 - Silhouette Coefficient Method to Find Best k

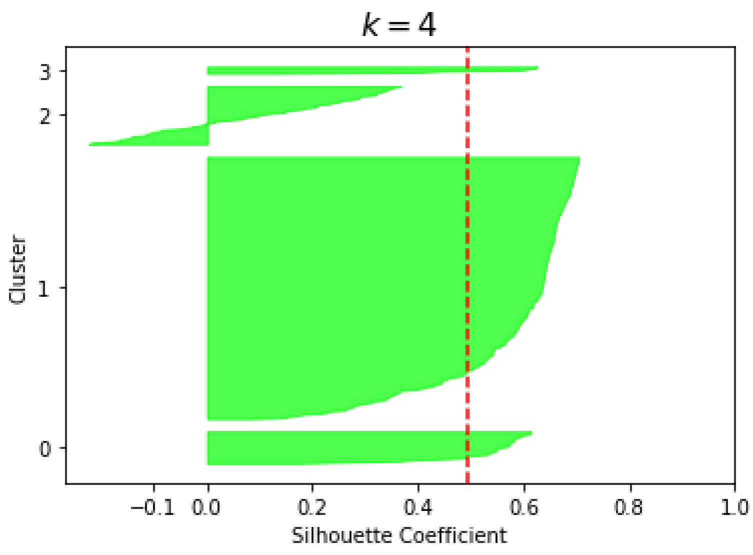
In [30]:

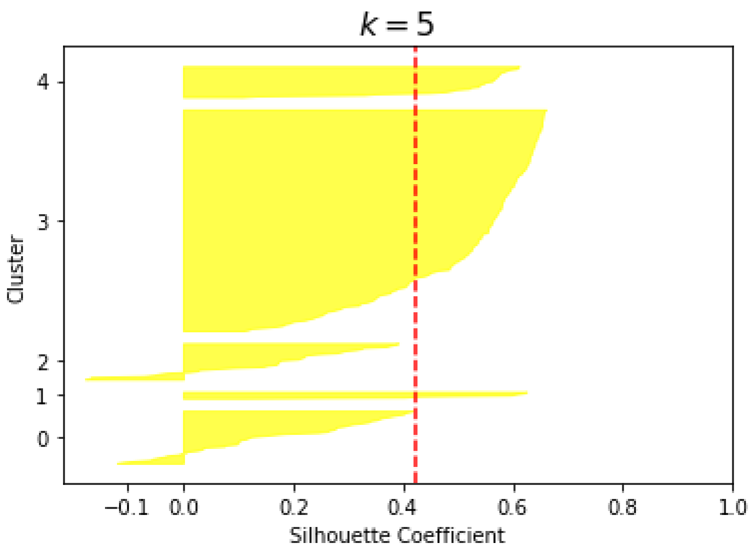
```
# From https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_ana
lysis.html
# From https://github.com/ageron/handson-ml2/blob/master/09_unsupervised_learning.ipynb

# Empty Array to Store Silhouette values
silhouette_avg = np.empty(len(kvals))


for i in range(len(kvals)):

    clustered_data_sklearn = cluster.KMeans(n_clusters=kvals[i],  n_init=n_init, max_it
er=max_iter, random_state=random_state).fit(scale_data)


    cluster_labels = clustered_data_sklearn.labels_

    # The silhouette_score gives the average value for all the samples.
    # This gives a perspective into the density and separation of the formed
    # clusters
    silhouette_avg[i] = silhouette_score(scale_data, cluster_labels)


    # Compute the silhouette scores for each sample
    sample_silhouette_values = silhouette_samples(scale_data, cluster_labels)

    padding = len(scale_data) // 30
    pos = padding
    ticks = []


    for j in range(kvals[i]):
        coeffs = sample_silhouette_values[cluster_labels==j]
        coeffs.sort()

        plt.fill_betweenx(np.arange(pos, pos + len(coeffs)), 0, coeffs,
                              facecolor=colors[i%colors.size], edgecolor=colors[i%colors.si
ze], alpha=0.7)

        ticks.append(pos + len(coeffs) // 2)
        pos += len(coeffs) + padding

    plt.gca().yaxis.set_major_locator(FixedLocator(ticks))
    plt.gca().yaxis.set_major_formatter(FixedFormatter(range(kvals[i])))
    plt.gca().set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])
    plt.xlabel("Silhouette Coefficient")
    plt.ylabel("Cluster")
    plt.axvline(x=silhouette_avg[i], color="red", linestyle="--")
    plt.title("$k={}$".format(kvals[i]), fontsize=16)
    plt.show()


# Plot Silhoette Scores

plt.figure(figsize=(8, 3))
plt.plot(kvals,  silhouette_avg, "bo-")
plt.xlabel("$k$", fontsize=14)
plt.ylabel("Silhouette score", fontsize=14)
plt.show()
```

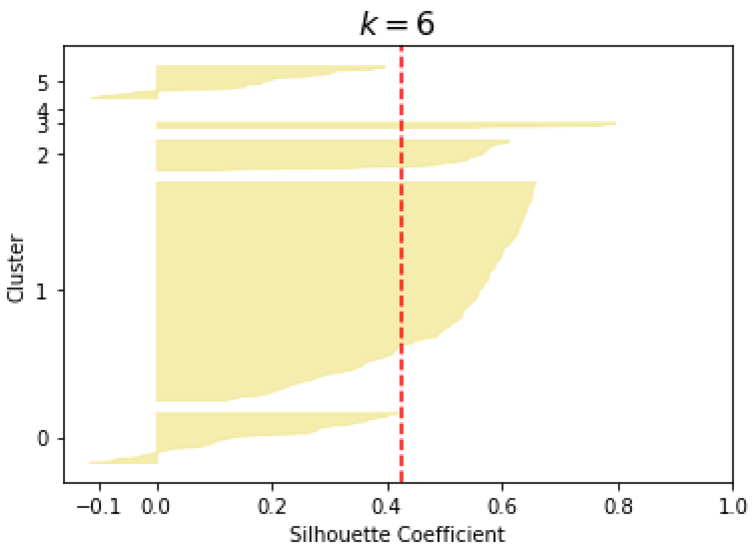
```
for i in range(len(kvals)):
    print("For k =", kvals[i], "The average silhouette_score is :", silhouette_avg[i])
```

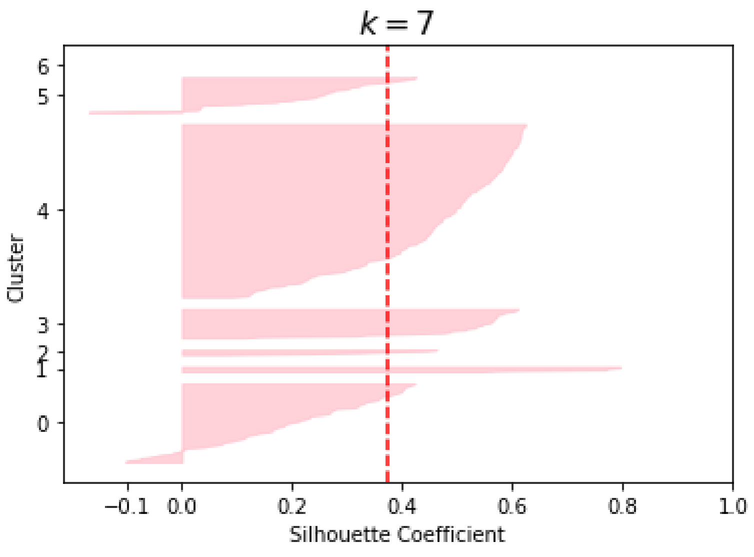

$k = 2$
Cluster
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
Silhouette Coefficient

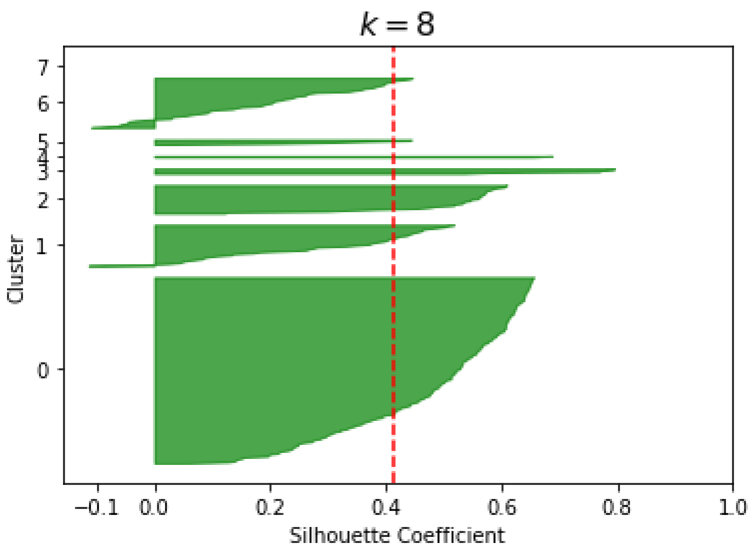

$k = 3$
Cluster
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
Silhouette Coefficient

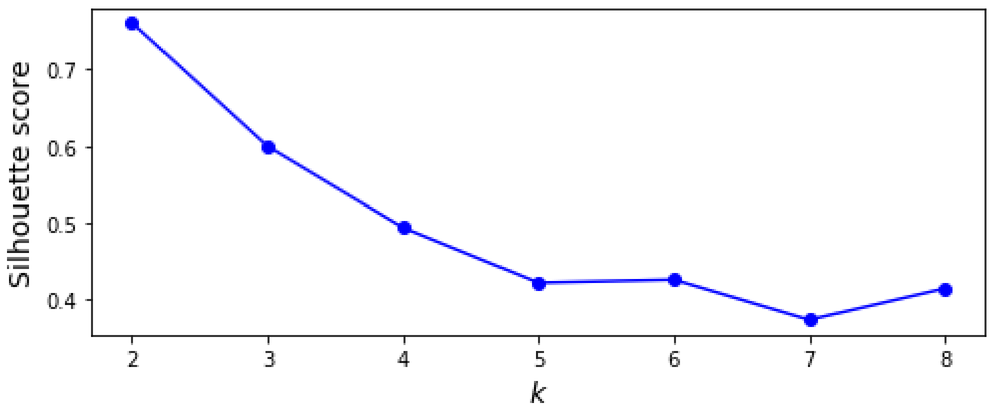

$k = 4$
Cluster
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
Silhouette Coefficient


$k = 5$
Cluster
Silhouette Coefficient
0
1
2
3
4
−0.1
0.0
0.2
0.4
0.6
0.8
1.0


$k = 6$
Cluster
Silhouette Coefficient
0
1
2
3
4
5
−0.1
0.0
0.2
0.4
0.6
0.8
1.0


$k = 7$
Cluster
Silhouette Coefficient
0
1
2
3
4
5
6
−0.1
0.0
0.2
0.4
0.6
0.8
1.0





```
For k = 2 The average silhouette_score is : 0.7596807546208396
For k = 3 The average silhouette_score is : 0.5993683877939564
For k = 4 The average silhouette_score is : 0.49382717460168524
For k = 5 The average silhouette_score is : 0.4222843719671475
For k = 6 The average silhouette_score is : 0.42634925012082986
For k = 7 The average silhouette_score is : 0.3741853059971289
For k = 8 The average silhouette_score is : 0.41493989203937404
```

SSE Plot, Elbow Plot and Silhouette Score Analysis

My initial value used k=4. With this value, one of the clusters only had 7 members (all with Credit_History=0). The Elbow plot showed that the elbow position was in the range 3 to 5. This correlated with the best looking values of K in the SSE plot.

Using the Silhouette method, I could see that for k=4, not all clusters were above the average silhouette score. A similar observation existed for k=5. However for K=3, all the clusters were above the average silhouette score. k=3 also have the second highest silhouette score.

Choosing K=3 as the best value of K to use for running K-Means

Re-Run K-Means with K = 3

In [31]:

```
# K-Means Parameters
k = 3
```

In [32]:

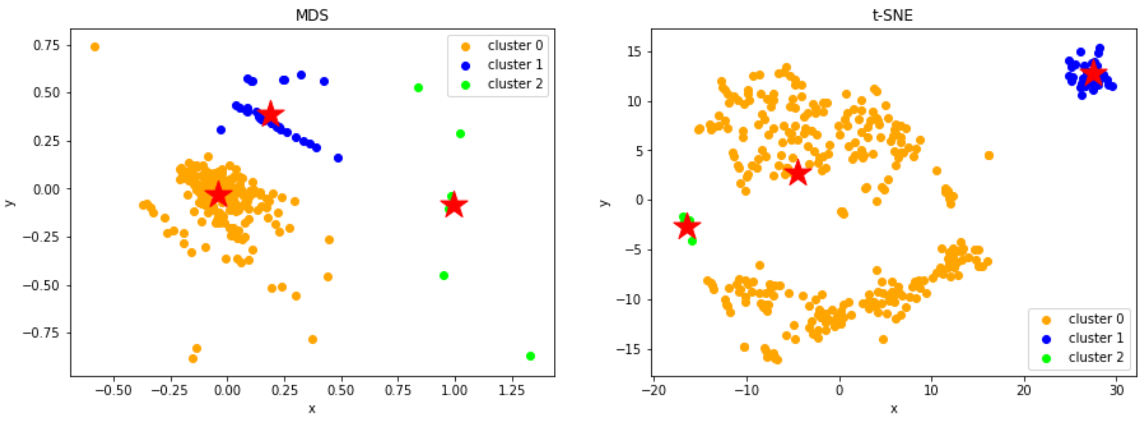
```
km_k_3 = cluster.KMeans(n_clusters=k, n_init=n_init, max_iter=max_iter, random_state=ra
ndom_state).fit(scale_data)
```

In [33]:

```
# append the cluster centers to the dataset
km_k_3_data_and_centers = np.r_[scale_data,km_k_3.cluster_centers_]
```

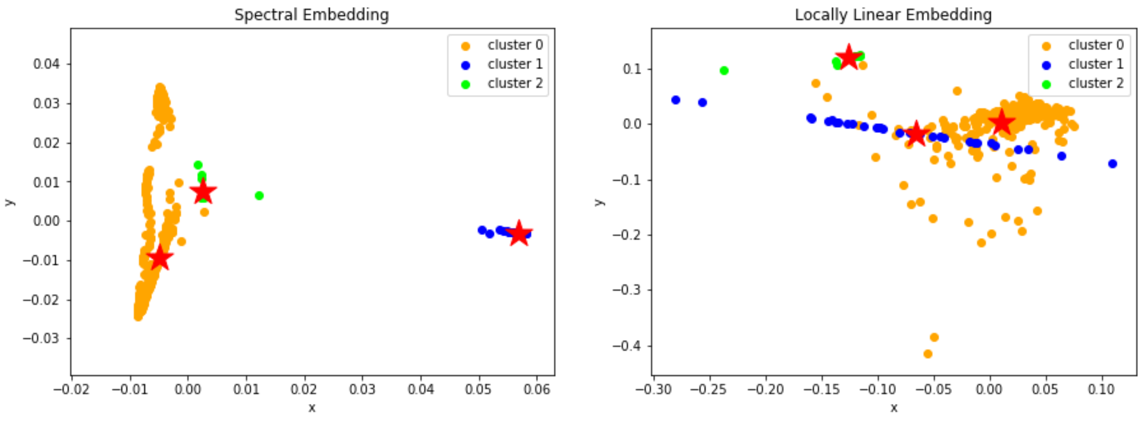
In [34]:

```
plot_mds_tsne(k, km_k_3_data_and_centers,km_k_3.labels_, n_components=2)
```



In [35]:

```
plot_se_lle(k, 10, km_k_3_data_and_centers,km_k_3.labels_, n_components=2)
```



MDS, t-SNE, SE and LLE Plot Analysis (*K*=3)

- In all plots, the k=3 Cluster 0 grouping looks like the k=4 Cluster 1 and Cluster 2 have been merged
- MDS, t-SNE and SE all have Cluster 1 (Short Term Loans) as very distinct clusters but LLE again plots this cluster as a subset of the other loadns
- t-SNE, SE and LLE all have Cluster 2(CreditHistory=0) as distinct clusters but interestingly MDS has this cluster a bit scattered which potentially shows that apart from all entries in this cluster having Credit History=0, the entries might be quite dissimilar in terms of other features

- Cluster 1 and Cluster 2 are very close in both plots
  - This suggests that elements might be similar
  - This will turn out to be true in later analysis
- Both plots have Cluster 3 (CreditHistory=0) and Cluster 0 (Smaller, Shorter Loans) as very distinct groupings

Task 2 - Cluster Description (k=3)

In [36]:

```
# Append the cluster labels to the original data
df['cluster_k3'] = pd.Series(km_k_3.labels_, index=df.index)
```

In [37]:

```
# Numer of Entries per Cluster
df['cluster_k3'].value_counts()
```

Out[37]:

```
0    295
1     30
2      7
Name: cluster_k3, dtype: int64
```

In [38]:

```
df.groupby('cluster_k3').mean()
```

Out[38]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_
cluster_k3					
0	5112.803390	1501.383458	141.827119	360.813559	
1	5146.966667	1297.126666	116.333333	161.600000	
2	9153.857143	2098.142857	206.285714	308.571429	

Task 2 - Cluster Characteristics (k=3)

By comparing the ***cluster_k3*** labels and the ***cluster_k4*** labels you can see that with k=3 it has merged Cluster 1 and Cluster 2 from the k=4 iteration into what is now Cluster 0 for k=3. The other two clusters have been preserved

This merged the previous "Low Risk, Average Income, Average Amount, Long Term Loan" and "Low Risk, High Income, High Amount, Long Term Loan" clusters suggesting the the amount and the income were not different enough to justify separate clusters. This new cluster is effectively "Low Risk, Long Term Loans"

Based on the above and learnings from K=4, the clusters can be characterised as follows:

- **Cluster 0:** Low Risk, Long Terms Loans
- **Cluster 1:** Low Risk, Average Income, Low Amount, Short Term Loan
- **Cluster 3:** High Risk, High Income, Medium Amount, Medium Term Loan

Task 2 - Additional Clustering Algorithm - Mean Shift

For the additional clustering algorithm, I have picked MeanShift based on what I read in Chris Albon's "Machine Learning with Python Cookbook". I thought it was interesting that this algorithm would automatically determine the number of clusters to use and to see how this compared against using K-means

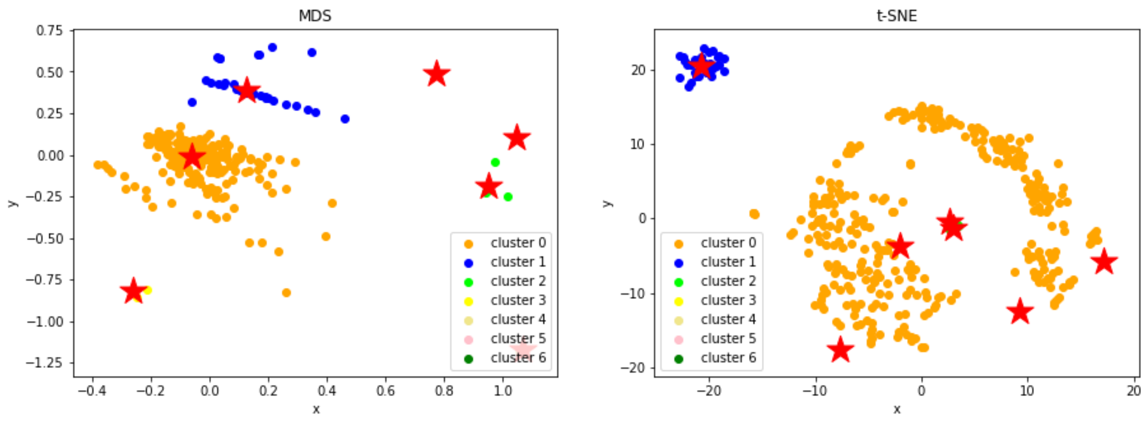
In [39]:

```
# https://learning.oreilly.com/library/view/machine-learning-with/9781491989371/ch19.ht
ml#clustering
# https://scikit-learn.org/stable/auto_examples/cluster/plot_mean_shift.html#sphx-glr-a
uto-examples-cluster-plot-mean-shift-py
from sklearn.cluster import MeanShift

# Set bandwidth
# If I just used MeanShift out of the box without changing any parameters, it broke the
data into 16 clusters.
# A lot of these clusters had only  single element and it was difficult to take any ins
ight from the visualizations
# or clusterings.
# Using trial and erro, the below settings appeared to work well
bandwidth = cluster.estimate_bandwidth(scale_data, quantile=0.8)

# Create meanshift object
#ms = MeanShift(bandwidth=bandwidth, bin_seeding=True, cluster_all=False)
ms = MeanShift(bandwidth=bandwidth, bin_seeding=True, cluster_all=True) # MPP -> Set to
true until visualisations debugged

# Train model
model = ms.fit(scale_data)

ms_labels = ms.labels_
ms_cluster_centers = ms.cluster_centers_

ms_labels_unique = np.unique(ms_labels)
ms_n_clusters_ = len(ms_labels_unique)

# append the cluster centers to the dataset
ms_scale_data_and_centers = np.r_[scale_data, ms.cluster_centers_]

print("number of estimated clusters : %d" % ms_n_clusters_)
```

```
number of estimated clusters : 7
```

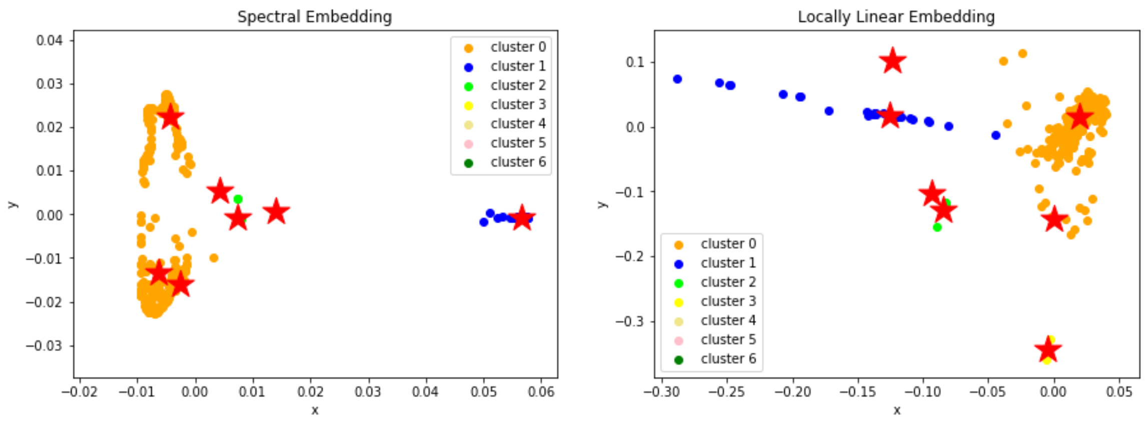
In [40]:

```
plot_mds_tsne(ms_n_clusters_,ms_scale_data_and_centers, ms.labels_, n_components=2)
```



In [41]:

```
plot_se_lle(ms_n_clusters_, 10, ms_scale_data_and_centers, ms.labels_, n_components=2)
```



Task 2 - Mean Shift Cluster Descriptions

In [42]:

```
# Append the cluster labels to the original data
df['cluster_ms'] = pd.Series(ms.labels_, index=df.index)
```

In [43]:

```
print(df['cluster_ms'].nunique())
df['cluster_ms'].value_counts()
```

```
7
```

Out[43]:

```
0    292
1     30
2      5
3      2
6      1
5      1
4      1
Name: cluster_ms, dtype: int64
```

In [44]:

```
df.groupby('cluster_ms').mean()
```

Out[44]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_Hi
cluster_ms					
0	4893.530822	1432.048356	142.000000	360.821918	
1	5146.966667	1297.126666	116.333333	161.600000	
2	4319.000000	2444.200000	150.800000	360.000000	
3	38433.000000	2375.000000	136.000000	360.000000	
4	39999.000000	0.000000	600.000000	180.000000	
5	2500.000000	20000.000000	103.000000	360.000000	
6	2483.000000	2466.000000	90.000000	180.000000	

K = 4:

- **Cluster 0:** Low Risk, Average Income, Low Amount, Short Term Loan (30 Entries)
- **Cluster 1:** Low Risk, Average Income, Average Amount, Long Term Loan (241 Entries)
- **Cluster 2:** Low Risk, High Income, High Amount, Long Term Loan (54 Entries)
- **Cluster 3:** High Risk, High Income, Medium Amount, Medium Term Loan (7 Entries)

K = 3:

- **Cluster 0:** Low Risk, Long Terms Loans (295 Entries)
- **Cluster 1:** Low Risk, Average Income, Low Amount, Short Term Loan (30 Entries)
- **Cluster 2:** High Risk, High Income, Medium Amount, Medium Term Loan (7 Entries)

From the "grouby" table we can see that Mean Shift Cluster 1 with 30 Entries is exactly equivalent to K=3 Cluster 1 in terms of the number of entries and the mean values of the attributes

- This cluster is ***Low Risk, Average Income, Low Amount, Short Term Loan***

Mean Shift Cluster 0 is the same as K=3 CLuster 0 except there are 3 fewer entries (295 vs 292). Of the 3 "missing entries", 2 are placed into Mean Shift Cluster 3 and one is placed into Mean Shift Cluster 5.

- Mean Shift Cluster 0 is ***Low Risk, Long Terms Loans***
- Mean Shift Cluster 3 has a mean Applicant Income of over 38K. These are effectively two outliers from Mean Shift Cluster 0
- Mean Shift Cluster 5 has a single entry that has a Coapplicant Income of 20K. This is the highest Coapplicant Income in the entire dataset and is also effectively an outlier from Mean Shift Cluster 0

Similarly, Mean Shift Cluster 2 is the same as K=3 CLuster 2 (Credit History=0) except there are 2 fewer entries. One of the missing entries is in Mean Shift Cluster 4 and the other is in Mean Shif Cluster 6

- Mean Shift Cluster 2 is ***High Risk, High Income, Medium Amount, Medium Term Loan***
- Mean Shift CLuster 4 has an ApplicantIncome of 39,999. This is the maximum ApplicantIncome in the entire dataset and is effectively an outlier from Mean Shift Cluster 2
- Mean Shift Cluster 6 has a combined income of less than 5K and got a loan even though Credit History was 0. This is an outlier from Mean Shift Cluster 2 as most examples with a Credit History of 0 had a combined income of over 11K. In effect, the high income made up for the poor credit history. The single example in this cluster is unusual in that it has a low income and a poor credit history and still got a loan.

Mean Shift 7 CLusters:

- **Cluster 0:** Low Risk, Long Terms Loans (292 Entries)
- **Cluster 1:** Low Risk, Average Income, Low Amount, Short Term Loan (30 Entries)
- **Cluster 2:** High Risk, High Income, Medium Amount, Medium Term Loan (5 Entries)
- **Cluster 3:** Very High Income, Low Risk, Long Terms Loans (2 Entries)
- **Cluster 4:** Very High Applicant Income, High Risk, High Income, Medium Amount, Medium Term Loan (1 Entries)
- **Cluster 5:** Very High Coapplicant Income, Low Risk, Long Terms Loans (1 Entries)
- **Cluster 6:** High Risk, Low Income, Medium Amount, Medium Term Loan (1 Entries)

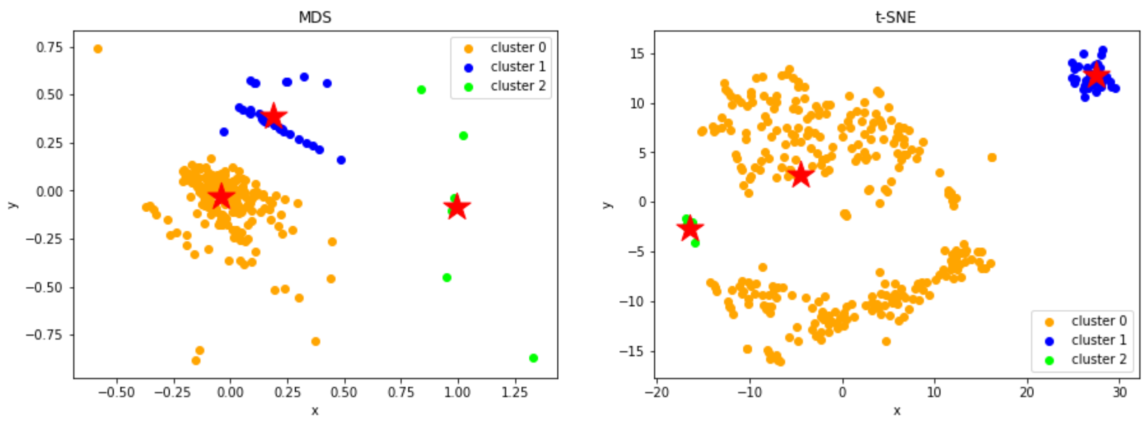
Task 2 - Visual Comparison of Clusterings

- Cluster 0 in both algorithms and both mappings is very similar and this is to be expected as it contains almost the same examples
- Cluster 1 in both algorithms is identical and the plottings reflect that similarity
- In the mean shift plots, you can see that the 4 additional clusters, relative to k=3, are used to group outliers that were contained within the original clusters for k = 3

In [45]:

```
print("K=3 K-Means Cluster")
plot_mds_tsne(k, km_k_3_data_and_centers,km_k_3.labels_, n_components=2)
```

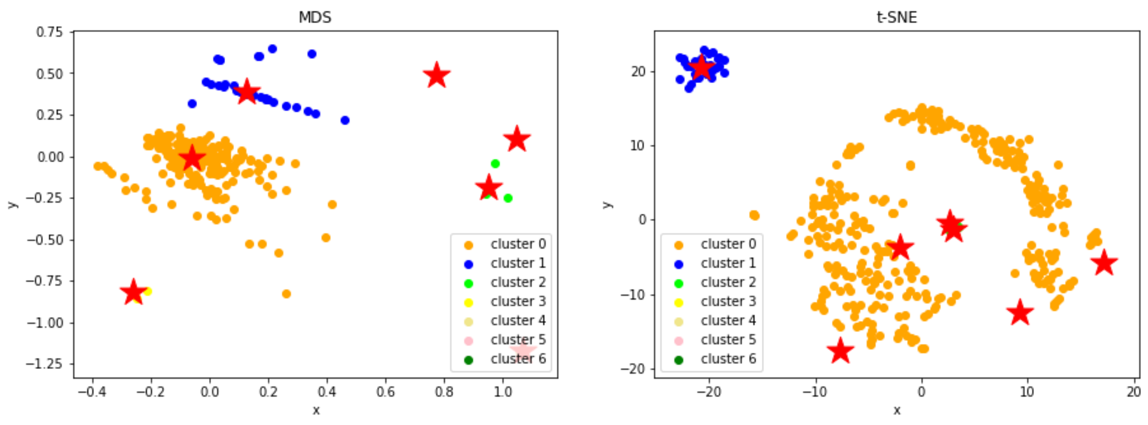
```
K=3 K-Means Cluster
```



In [46]:

```
print("7 Mean-Shift Clusters")
plot_mds_tsne(ms_n_clusters_,ms_scale_data_and_centers, ms.labels_, n_components=2)
```

```
7 Mean-Shift Clusters
```



Task 2 - Difference Between K-Means And Second Algorithm Discussion

The sections above discuss the differences between the K-means clusters and the Mean Shift clusters and have visualised the clusters using MDS and t-SNE.

The key difference between the algorithms is that Mean Shift determines the numbers of clusters required. The number of clusters that Mean Shift chooses is a factor of the bandwidth parameter and this must be tuned in order to get meaningful clusters from Mean Shift. I found that with certain bandwidth values, I was getting up to 16 clusters and most of these clusters only had a single entry which was not very meaningful.

In the end I ended up with 7 clusters from Mean Shift and 3 of these clusters were almost identical to the corresponding clusters obtained when running K-Means with k=3. The additional four clusters very effectively outliers from within the 3 main clusters. In this sense, Mean Shift seemed to perform a filtering operation on the extreme values within the main clusters.

Mean Shift Algorithm Description

From Chris Albon's "Machine Learning with Python Cookbook":

Meanshift is a simple concept, but somewhat difficult to explain. Therefore, an analogy might be the best approach. Imagine a very foggy football field (i.e., a two-dimensional feature space) with 100 people standing on it (i.e., our observations). Because it is foggy, a person can only see a short distance. Every minute each person looks around and takes a step in the direction of the most people they can see. As time goes on, people start to group up as they repeatedly take steps toward larger and larger crowds. The end result is clusters of people around the field. People are assigned to the clusters in which they end up.

scikit-learn’s actual implementation of meanshift, MeanShift, is more complex but follows the same basic logic. MeanShift has two important parameters we should be aware of. First, bandwidth sets the radius of the area (i.e., kernel) an observation uses to determine the direction to shift. In our analogy, bandwidth was how far a person could see through the fog. We can set this parameter manually, but by default a reasonable bandwidth is estimated automatically (with a significant increase in computational cost). Second, sometimes in meanshift there are no other observations within an observation’s kernel. That is, a person on our football field cannot see a single other person. By default, MeanShift assigns all these “orphan” observations to the kernel of the nearest observation. However, if we want to leave out these orphans, we can set cluster_all=False wherein orphan observations are given the label of -1.