UNIVERSITY OF MALAYA

EXAMINATION FOR THE DEGREE OF MASTER OF DATA SCIENCE

ACADEMIC SESSION 2019/2020 : SEMESTER II

WQD7005 : Data Mining

June 2020

INSTRUCTIONS TO CANDIDATES :

Answer **ALL** questions (50 marks).

Name : Nurullainy Binti Mat Rashid
ID : 17036591

Link to codes and data:
https://github.com/Nurullainy/Data-Mining-Final-Exam

Link to video :
https://drive.google.com/drive/folders/1hMqLYw5ubSpMRsPvyJwjDG8MuV3GO_Qt?usp=sharing

(This question paper consists of 5 questions on 3 printed pages)

Mini-assignment (50 marks)
Instructions: Work individually, submission via Spectrum.

. 1. You are required to make a user-agent that will crawl the WWW (your familiar domain) to produce dataset of a particular website.
- the web site can be as simple as a list of webpages and what other pages they link to
- the output does not need to be in XHTML (or HTML) form
  a multi-stage approach (e.g. produce the xhtml or html in csv format)

(10 marks)

2. Draw snowflake schema diagram for the above dataset. Justify your attributes to be selected in the respective dimensions.

(10 marks)

3. You are required to write code to create a decision tree (DT) model using the above dataset (Question 1). In order to achieve the task, you are going to cover the following steps:
- Importing required libraries
- Loading Data
- Feature Selection
- Splitting Data
- Building Decision Tree Model
- Evaluating Model
- Visualizing Decision Trees

(10 marks)

4. You are required to write code to find frequent itemsets using the above dataset (Question 1). In order to achieve the task, you are going to cover the following steps:
- Importing required libraries
- Creating a list from dataset (Question 1)
- Convert list to dataframe with boolean values
- Find frequently occurring itemsets using Apriori Algorithm
- Find frequently occurring itemsets using F-P Growth
- Mine the Association Rules

(10 marks)

5. You are required to write code to implement either time-series clustering or density-based clustering model using the above dataset (Question 1). If you select density-based clustering approach to achieve the task, you are going to cover the following steps:

I have created 2 density-based clustering model. The first model is with application of Principal Component Analysis (PCA) and the second model is without application of PCA. I would like to view the difference between these 2 clustering models.

5a) Density-Based Clustering with PCA of Internet Movie Database (IMDb)

- Importing required libraries

```
In [73]: import pandas as pd
         import numpy as np

         import seaborn as sns
         import matplotlib.pyplot as plt
         from sklearn.decomposition import PCA
         from sklearn import preprocessing
         from sklearn.preprocessing import StandardScaler
         from sklearn.cluster import DBSCAN
         import sklearn.utils
         from sklearn.neighbors import NearestNeighbors

         import math
```

- Load the dataset (Question 1) into a DataFrame object

```
In [74]: # Load dataset

         df = pd.read_csv('movies_imdb_preprocessed.csv')
         df.head()
```

Out[74]:

	movie_name	year_released	runtime_in_min	genre	revenues	imdb_rating	user_votes	director	actor
0	Gladiator	2000	155	Action, Adventure, Drama	187705427	8.5	1295546	Ridley Scott	Russell Crowe, Joaquin Phoenix, Connie Nielsen...
1	Memento	2000	113	Mystery, Thriller	25544867	8.4	1088700	Christopher Nolan	Guy Pearce, Carrie-Anne Moss, Joe Pantoliano, ...
2	Snatch	2000	104	Comedy, Crime	30328156	8.3	760646	Guy Ritchie	Jason Statham, Brad Pitt, Benicio Del Toro, De...
3	Requiem for a Dream	2000	102	Drama	3635482	8.3	742193	Darren Aronofsky	Ellen Burstyn, Jared Leto, Jennifer Connelly, ...
4	X-Men	2000	104	Action, Adventure, Sci-Fi	157299717	7.4	558716	Bryan Singer	Patrick Stewart, Hugh Jackman, Ian McKellen, F...

```
In [75]: df.isnull().sum()
```

```
Out[75]: movie_name         0
         year_released      0
         runtime_in_min     0
         genre              0
         revenues           0
         imdb_rating        0
         user_votes         0
         director          78
         actor             78
         dtype: int64
```

```
In [76]: # Remove categorical data for PCA analysis

         df = df.drop(['movie_name', 'genre', 'director', 'actor'], axis=1)
```

```
In [77]: df.shape
```

```
Out[77]: (894, 5)
```

- Visualize the data, use only two of these attributes at the time

  Apply PCA to reduce 5 dimensional data into 2 dimensions for better viewing of data contained in a dataset. In summary, PCA finds and eliminate less informative (duplicate) information on feature set and reduce the dimension of feature space.

In [78]:
```
# Performs PCA computations into 2 dimensions

pca = PCA(n_components = 2, whiten = False, random_state = 0)

data_pca = pca.fit_transform(df)
```

In [79]:
```
data_pca.shape
```

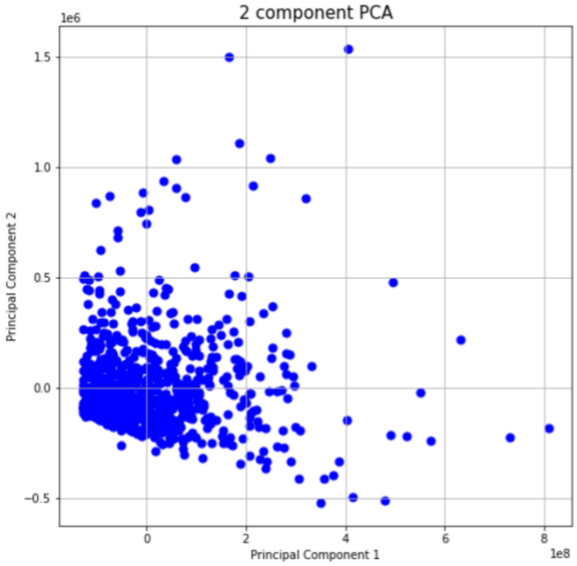
Out[79]:
```
(894, 2)
```

In [80]:
```
df2 = pd.DataFrame(data = data_pca, columns = ['principal component 1', 'principal component 2'])
df2.head()
```

Out[80]:

	principal component 1	principal component 2
0	5.961064e+07	906207.197060
1	-1.025500e+08	837179.681135
2	-9.776702e+07	505060.536728
3	-1.244597e+08	509293.342529
4	2.920432e+07	195218.928241

In [81]:
```
# Visualize 2 Dimensional data

fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)

ax.scatter(df2.loc[:, 'principal component 1'],
               df2.loc[:, 'principal component 2'],
               c = 'blue',
               s = 50)

ax.set_xlabel('Principal Component 1', fontsize = 10)
ax.set_ylabel('Principal Component 2', fontsize = 10)
ax.set_title('2 component PCA', fontsize = 15)
ax.grid()
```



From 5 features set, we are now dealing with only 2 features. These newly created features are also known as Principal Components (PC).

The above plot tells us that the observations in the dataset can be grouped. Each data points in the data is a representing one movie. We could say that the clusters represent different movie. This dataset does not have a target variable by which to label these groups, so we do not know exactly what these labels are.

To visualize the reduced dataset with much greater granularity, I will perform density-based clustering in later section.

- You may need to normalise the attribute if necessary

```
In [82]: # Standardizing the features

         normalized_data = StandardScaler().fit_transform(df2)
```

```
In [83]: df3 = pd.DataFrame(data = normalized_data, columns = ['principal component 1', 'principal component 2'])
         df3.head()
```

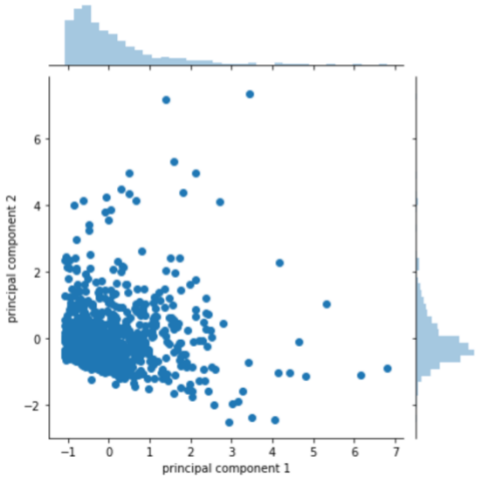
Out[83]:

	principal component 1	principal component 2
0	0.502186	4.343581
1	-0.863926	4.012722
2	-0.823632	2.420828
3	-1.048503	2.441116
4	0.246030	0.935712

- Show positive correlation between attributes if necessary

```
In [84]: # Another data visualization using jointplot

         sns.jointplot(x= "principal component 1", y="principal component 2", data=df3)
```

```
Out[84]: <seaborn.axisgrid.JointGrid at 0x11e857d30>
```



A positive correlation is a relationship between two variables in which both variables move in the same direction. Therefore, when one variable increases as the other variable increases, or one variable decreases while the other decreases. In this case, the above plot shows there is positive correlation between IMDb rating and revenue generated from each movie. Movie with high revenue has high IMDb rating at the same time.

- Construct a density-based clustering model and extract cluster labels and outliers to plot your results.

eps is the maximum distance between two points. It is this distance that the algorithm uses to decide on whether to group the two points together.

min_samples is minimum number of neighbors a given point should have in order to be classified as a core point.

The algorithm will starts with an arbitrary point (p point) and retrieve all points density-reachable from the p point with respect to eps and min_samples sets in the algorithm. If p point is a core point, it will result a cluster and if p point is a border point, no points are density-reachable from p point. Then the model visits the next point in the database.

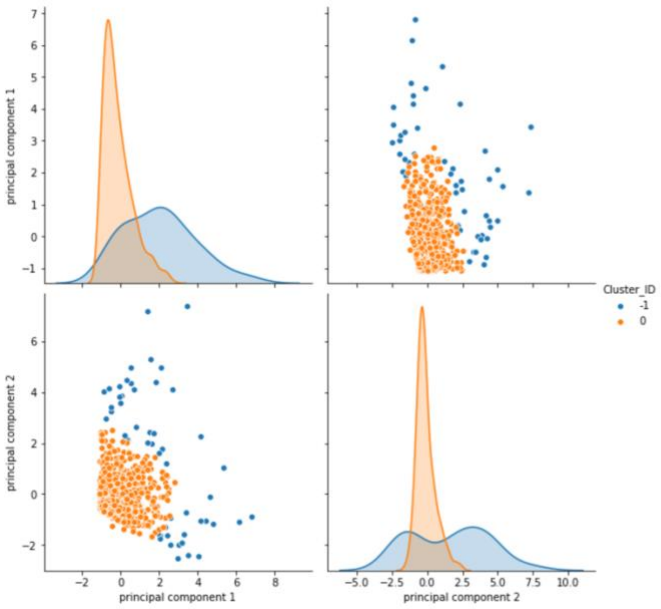
DBSCAN works by determining whether the minimum number of points are close enough to one another to be considered part of a single cluster or otherwise. In simpler word, DBSCAN finds density-connected regions.

In [85]:
```
# Construct a density-based clustering model

db = DBSCAN(eps=0.5, min_samples=10).fit(df3)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
```

In [86]:
```
# Extract cluster labels and outliers

labels = db.labels_
df3['Cluster_ID'] = labels
```

In [87]:
```
# Number of records are in each cluster
print("Number of Cluster:", len(df3['Cluster_ID'].value_counts()))
print("Cluster membership")
print(df3['Cluster_ID'].value_counts())

# Pairplot the cluster distribution.
cluster_db = sns.pairplot(df3, hue='Cluster_ID', height=4)
plt.show()
```

```
Number of Cluster: 2
Cluster membership
 0    842
-1     52
Name: Cluster_ID, dtype: int64
```



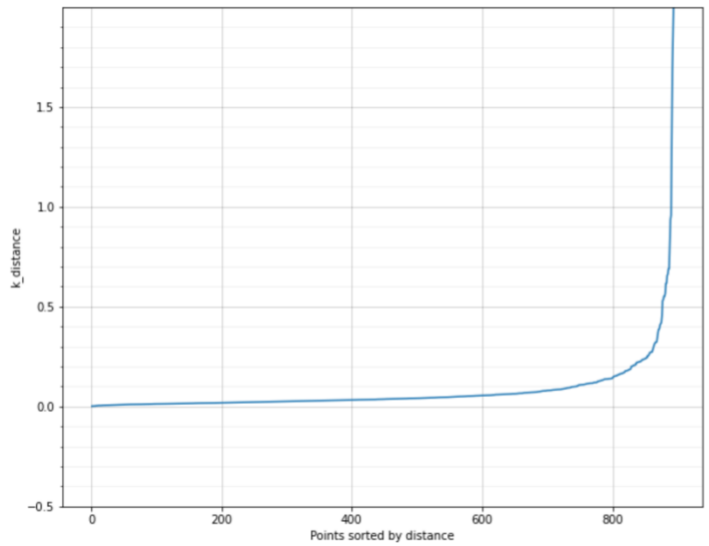
Number of cluster identified by the model is 2

Finding optimal epsilon value using k-distance

There is no general way of choosing minPts. However, a low minPts means it will build more clusters from noise, hence I maintain min_sample as 10.

I will calculate the distances of every point to its closest neighbour (k-distances) using the NearestNeighbors from sklearn library. The algorithm works by computing the distance between every point and all other points. These k-distances are then plotted in ascending order. The point where an elbow like bend corresponds to the optimal eps value.
metric is the metric to use when calculating distance between instances in a feature array (i.e. euclidean distance).

In [88]:
```
# Find k-distance and plot in ascending order

ns = 2
nbrs = NearestNeighbors(n_neighbors=ns, metric='euclidean').fit(df3)
distances, indices = nbrs.kneighbors(df3)
k_distance = sorted(distances[:,ns-1], reverse=False)    # sort the distance


fig, ax = plt.subplots(figsize=(10, 8))
plt.plot(list(range(1,len(df3)+1)), k_distance)
plt.axis([None, None, -0.5, 2])
plt.xlabel ('Points sorted by distance')
plt.ylabel ('k_distance')

major_ticks = np.arange(-0.5, 2, 0.5)
minor_ticks = np.arange(-0.5, 2, 0.1)
ax.set_yticks(major_ticks)
ax.set_yticks(minor_ticks, minor=True)
ax.grid(which='minor', alpha=0.2)
ax.grid(which='major', alpha=0.2, color='black')
```



The above graph shows a sharp change in the distance occurs at k_distance=0.15, and thus this point serves as a threshold. 0.15 is the optimal epsilon value for this data.

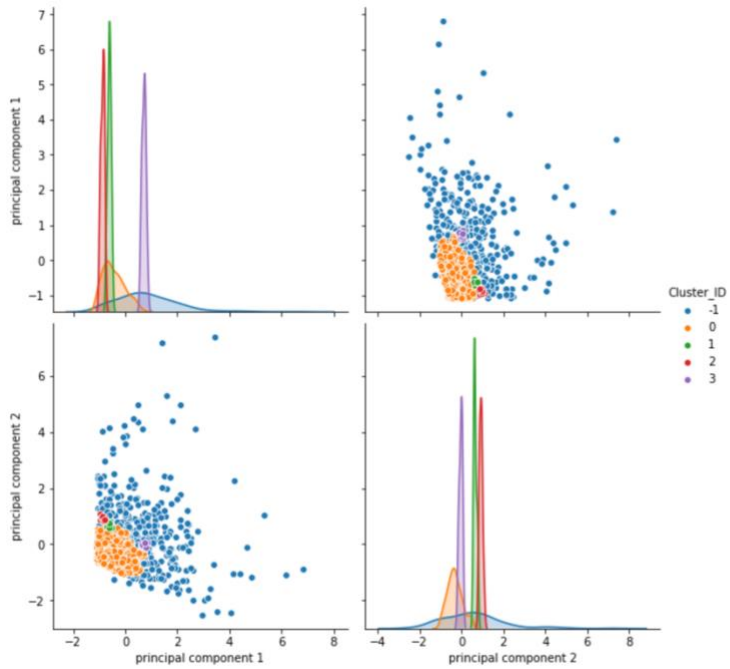
Construct a Density-Based Clustering model with optimal epsilon value, eps=0.15

In [89]:
```
# Construct a density-based clustering model

db = DBSCAN(eps=0.15, min_samples=10).fit(df3)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
```

In [90]:
```
# Extract cluster labels and outliers

labels = db.labels_
df3['Cluster_ID'] = labels
```

In [91]:
```
# Number of records are in each cluster
print("Number of Cluster:", len(df3['Cluster_ID'].value_counts()))
print("Cluster membership")
print(df3['Cluster_ID'].value_counts())

# Pairplot the cluster distribution.
cluster_db = sns.pairplot(df3, hue='Cluster_ID', height=4)
plt.show()
```

```
Number of Cluster: 5
Cluster membership
 0    570
-1    294
 1     10
 3     10
 2     10
Name: Cluster_ID, dtype: int64
```



Based on optimal epsilon value, number of cluster identified by the model is 5. Unlike k-means, DBSCAN will figure out the number of clusters in the dataset. DBSCAN is very sensitive to scale since epsilon is a fixed value for the maximum distance between two points. Different epsilon value yield different number of cluster. In this case, eps= 0.5 yield 2 clusters however if eps=0.15, the output is 5 clusters.

5b) Density-Based Clustering of Internet Movie Database (IMDb) – without PCA

- Importing required libraries

In [73]:
```
import pandas as pd
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import DBSCAN
import sklearn.utils
from sklearn.neighbors import NearestNeighbors

import math
```

- Load the dataset (Question 1) into a DataFrame object

In [74]:
```
# Load dataset

df = pd.read_csv('movies_imdb_preprocessed.csv')
df.head()
```

Out[74]:

	movie_name	year_released	runtime_in_min	genre	revenues	imdb_rating	user_votes	director	actor
0	Gladiator	2000	155	Action, Adventure, Drama	187705427	8.5	1295546	Ridley Scott	Russell Crowe, Joaquin Phoenix, Connie Nielsen...
1	Memento	2000	113	Mystery, Thriller	25544867	8.4	1088700	Christopher Nolan	Guy Pearce, Carrie-Anne Moss, Joe Pantoliano, ...
2	Snatch	2000	104	Comedy, Crime	30328156	8.3	760646	Guy Ritchie	Jason Statham, Brad Pitt, Benicio Del Toro, De...
3	Requiem for a Dream	2000	102	Drama	3635482	8.3	742193	Darren Aronofsky	Ellen Burstyn, Jared Leto, Jennifer Connelly, ...
4	X-Men	2000	104	Action, Adventure, Sci-Fi	157299717	7.4	558716	Bryan Singer	Patrick Stewart, Hugh Jackman, Ian McKellen, F...

In [75]:
```
df.isnull().sum()
```

Out[75]:
```
movie_name         0
year_released      0
runtime_in_min     0
genre              0
revenues           0
imdb_rating        0
user_votes         0
director          78
actor             78
dtype: int64
```

In [76]:
```
# Remove categorical data for PCA analysis

df = df.drop(['movie_name', 'genre', 'director', 'actor'], axis=1)
```

In [77]:
```
df.shape
```

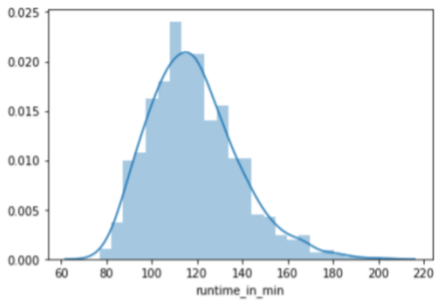
Out[77]:
```
(894, 5)
```

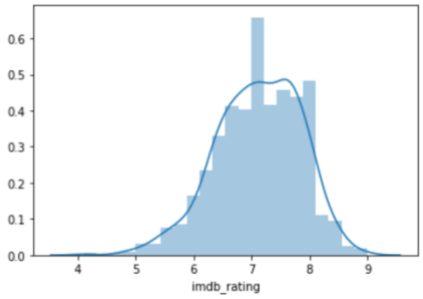
- Visualize the data, use only two of these attributes at the time

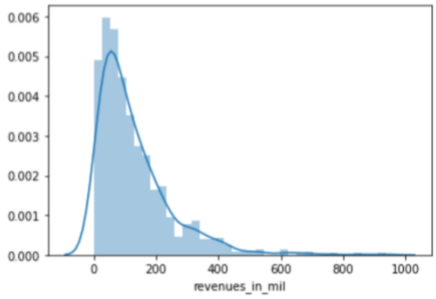
  Visualisation is a great way to spot data problems within the dataset. Again, we will use seaborn and matplotlib for that purpose. Plot the distribution of the variables using distplot.

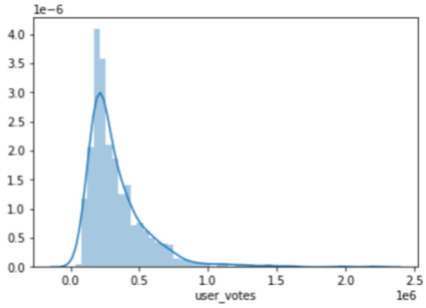
In [16]:
```
# Change million notation for data visualization

df_revenue = df['revenues'].div(1000000).to_frame('col') # Change to Million notation
df_revenue.shape

df['revenues_in_mil'] = df_revenue['col']

df['revenues_in_mil'] = df['revenues_in_mil'].round(0).astype(int)
```

In [17]:
```
# Distribution of runtime
runtime_dist = sns.distplot(df['runtime_in_min'].dropna())
plt.show()

# Distribution of revenues
revenue_dist = sns.distplot(df['revenues_in_mil'].dropna())
plt.show()

# Distribution of imdb_rating
rating_dist = sns.distplot(df['imdb_rating'].dropna())
plt.show()

# Distribution of user_votes
votes_dist = sns.distplot(df['user_votes'].dropna())
plt.show()
```



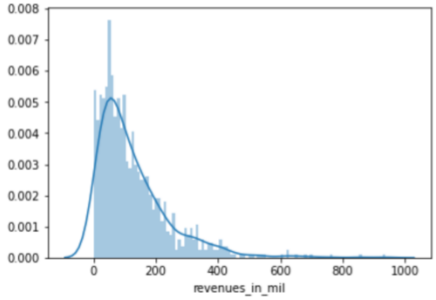






The above attributes showed normal distribution. We also could "zoom in" on the distribution by increasing the number of bins. View revenues distribution with more number of bin.

In [18]:
```
# Distribution of imdb ratings, with increased number of bins. More bins = more specific distplot.

rating_dist = sns.distplot(df['revenues_in_mil'].dropna(), bins=100)
plt.show()
```



Use only two of these attributes at the time

There are a number of good grouping objectives that can be applied in this dataset. I could cluster them based on runtime, revenue, user votes or imdb ratings. For this case, I would like to focus on revenues and see the combination from other attributes such as movie's runtime.

Revenue and Movie Runtime Clustering

In [19]:
```
revenue_runtime = df[['revenues', 'runtime_in_min', ]]

revenue_runtime.head()
```

Out[19]:

	revenues	runtime_in_min
0	187705427	155
1	25544867	113
2	30328156	104
3	3635482	102
4	157299717	104

- You may need to normalise the attribute if necessary

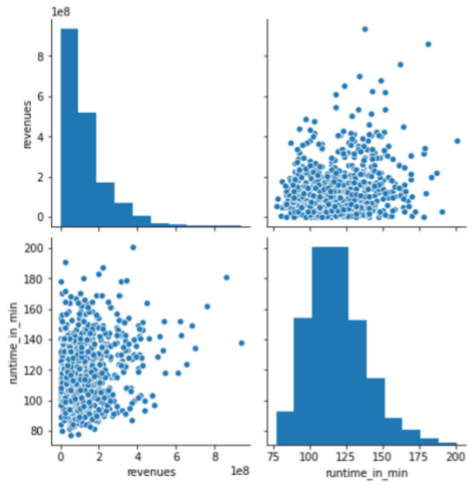
Because some of the values are in thousands and millions, I need to normalize each attribute by scaling it to 0 mean and unit variance.

In [20]:
```
data = revenue_runtime.as_matrix().astype("float32", copy = False)

stscaler = StandardScaler().fit(data)
data = stscaler.transform(data)
```

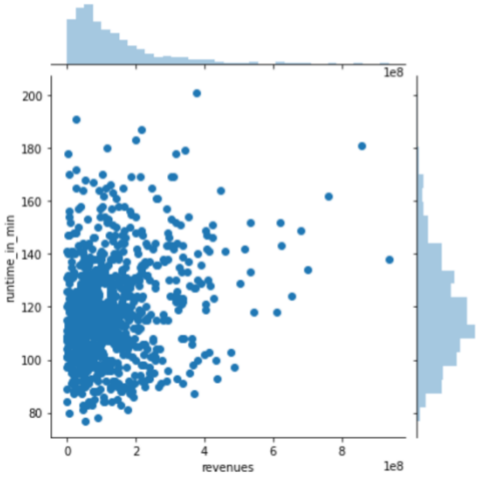
```
/usr/local/lib/python3.6/site-packages/ipykernel_launcher.py:1: FutureWarning: Method .as_matrix will be removed in
a future version. Use .values instead.
  """Entry point for launching an IPython kernel.
```

- Show positive correlation between attributes if necessary

In [21]:
```
# Pairplot the data distribution.

fg1 = sns.pairplot(revenue_runtime, height=3)
plt.show()
```



```
In [22]: # Another data visualization using jointplot

         sns.jointplot(x="revenues", y="runtime_in_min", data=revenue_runtime)

Out[22]: <seaborn.axisgrid.JointGrid at 0x126554860>
```



A positive correlation is a relationship between two variables in which both variables move in the same direction. Therefore, when one variable increases as the other variable increases, or one variable decreases while the other decreases. In this case, the above plot shows there is positive correlation between movie runtime and revenue generated from each movie. Movie with high revenue has high runtime at the same time.

- Construct a density-based clustering model and extract cluster labels and outliers to plot your results.

  `eps` is the maximum distance between two points. It is this distance that the algorithm uses to decide on whether to group the two points together.

  `min_samples` is minimum number of neighbors a given point should have in order to be classified as a core point.

  The algorithm will starts with an arbitrary point (p point) and retrieve all points density-reachable from the p point with respect to `eps` and `min_samples` sets in the algorithm. If p point is a core point, it will result a cluster and if p point is a border point, no points are density-reachable from p point. Then the model visits the next point in the database.

  DBSCAN works by determining whether the minimum number of points are close enough to one another to be considered part of a single cluster or otherwise. In simpler word, DBSCAN finds density-connected regions.

Construct a Density-Based Clustering model using DBSCAN function with eps = 0.5

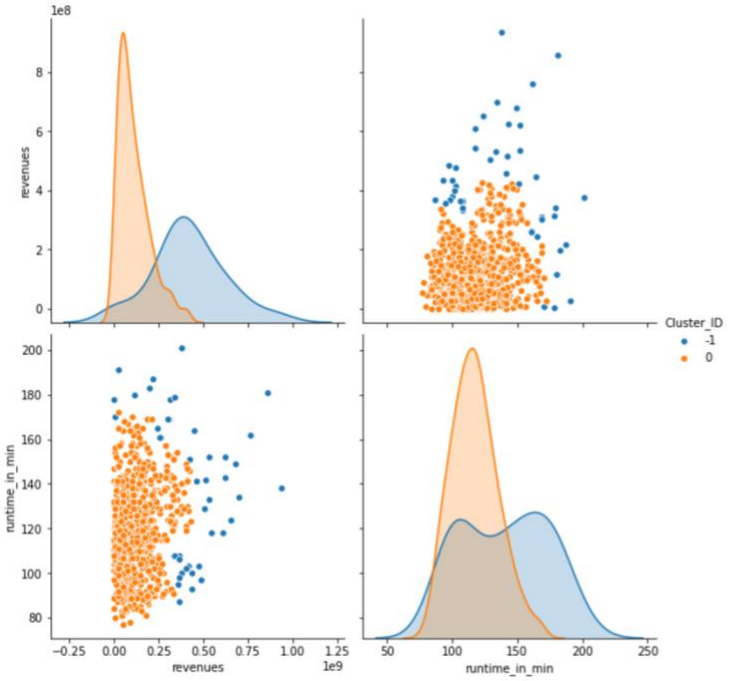
In [23]:
```
# Construct a density-based clustering model

db = DBSCAN(eps=0.5, min_samples=10).fit(data)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
```

In [24]:
```
# Extract cluster labels and outliers

labels = db.labels_
revenue_runtime['Cluster_ID'] = labels
```

```
/usr/local/lib/python3.6/site-packages/ipykernel_launcher.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  after removing the cwd from sys.path.
```

In [25]:
```
# Number of records are in each cluster
print("Number of Cluster:", len(revenue_runtime['Cluster_ID'].value_counts()))
print("Cluster membership")
print(revenue_runtime['Cluster_ID'].value_counts())

# Pairplot the cluster distribution.
cluster_db = sns.pairplot(revenue_runtime, hue='Cluster_ID', height=4)
plt.show()
```

```
Number of Cluster: 2
Cluster membership
 0    849
-1     45
Name: Cluster_ID, dtype: int64
```



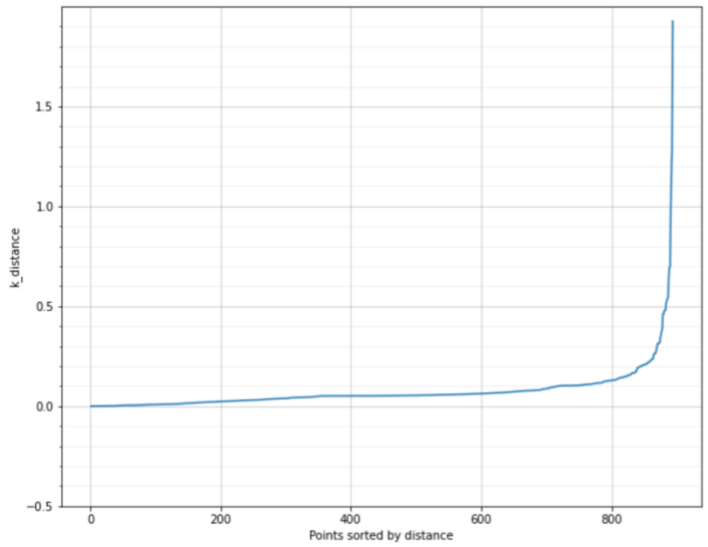
Number of cluster identified by the model is 2

Finding optimal epsilon value using k-distance

There is no general way of choosing minPts. However, a low minPts means it will build more clusters from noise, hence I maintain min_sample as 10.

I will calculate the distances of every point to its closest neighbour (k-distances) using the NearestNeighbors from sklearn library. The algorithm works by computing the distance between every point and all other points. These k-distances are then plotted in ascending order. The point where an elbow like bend corresponds to the optimal eps value.
metric is the metric to use when calculating distance between instances in a feature array (i.e. euclidean distance).

In [26]:
```
# Find k-distance and plot in ascending order

ns = 2
nbrs = NearestNeighbors(n_neighbors=ns, metric='euclidean').fit(data)
distances, indices = nbrs.kneighbors(data)
k_distance = sorted(distances[:,ns-1], reverse=False)    # sort the distance


fig, ax = plt.subplots(figsize=(10, 8))
plt.plot(list(range(1,len(data)+1)), k_distance)
plt.axis([None, None, -0.5, 2])
plt.xlabel ('Points sorted by distance')
plt.ylabel ('k_distance')

major_ticks = np.arange(-0.5, 2, 0.5)
minor_ticks = np.arange(-0.5, 2, 0.1)
ax.set_yticks(major_ticks)
ax.set_yticks(minor_ticks, minor=True)
ax.grid(which='minor', alpha=0.2)
ax.grid(which='major', alpha=0.2, color='black')
```



The above graph shows a sharp change in the distance occurs at k_distance=0.12, and thus this point serves as a threshold. 0.15 will be optimal epsilon value for this data.

Construct a Density-Based Clustering model with optimal epsilon value, eps=0.12

Construct a Density-Based Clustering model with optimal epsilon value, eps=0.12

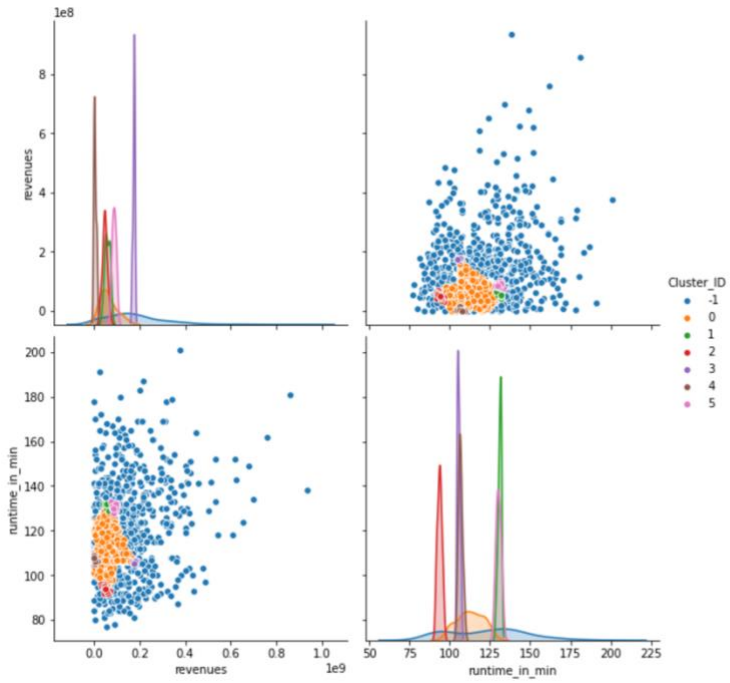
In [27]:
```
# Construct a density-based clustering model

db = DBSCAN(eps=0.12, min_samples=10).fit(data)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
```

In [28]:
```
# Extract cluster labels and outliers

labels = db.labels_
revenue_runtime['Cluster_ID'] = labels
```

```
/usr/local/lib/python3.6/site-packages/ipykernel_launcher.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  after removing the cwd from sys.path.
```

In [29]:
```
# Number of records are in each cluster
print("Number of Cluster:", len(revenue_runtime['Cluster_ID'].value_counts()))
print("Cluster membership")
print(revenue_runtime['Cluster_ID'].value_counts())

# Pairplot the cluster distribution.
cluster_db = sns.pairplot(revenue_runtime, hue='Cluster_ID', height=4)
plt.show()
```

```
Number of Cluster: 7
Cluster membership
-1    488
 0    353
 2     14
 1     13
 5     11
 4      9
 3      6
Name: Cluster_ID, dtype: int64
```



Based on optimal epsilon value, number of cluster identified by the model is 7. Unlike k-means, DBSCAN will figure out the number of clusters in the dataset. DBSCAN works by determining whether the minimum number of points are close enough to one another to be considered part of a single cluster or otherwise.

DBSCAN is very sensitive to scale since epsilon is a fixed value for the maximum distance between two points.

(10 marks)

Submissions:

The student is expected to submit answers to each question individually, and submit the document in PDF format. The student can include online materials, screenshots, videos and/or codes (ipynb format) to support your answer