In [ ]:

```
##PRIVACY PRESERVING ALGORITHMS IN MACHINE LEARNING###
#Privacy-Preserving Machine Learning is a step-by-step approach to preventing data leakag
e in machine learning algorithms.
#Referred From- " https://www.analyticsvidhya.com/blog/2022/02/privacy-preserving-in-mach
ine-learning-ppml/#:~:text=These%20techniques%20include%20perturbation%20techniques,speci
fic%20approaches%20like%20federated%20learning."
```

In [ ]:

```
#Diffprivlib is a general-purpose library for experimenting with, investigating and devel
oping applications in, differential privacy.

#Use diffprivlib if you are looking to:

# - Experiment with differential privacy
# - Explore the impact of differential privacy on machine learning accuracy using classif
ication and clustering models
# - Build your own differential privacy applications, using our extensive collection of m
echanisms
# Please Refer the reference folder in LAB-13 -  Paper - "Diffprivlib: The IBM Dierential
Privacy Library"
```

Diffprivlib is comprised of four major components:

Mechanisms: These are the building blocks of differential privacy, and are used in all models that implement differential privacy. Mechanisms have little or no default settings, and are intended for use by experts implementing their own models. They can, however, be used outside models for separate investigations, etc.

Models: This module includes machine learning models with differential privacy. Diffprivlib currently has models for clustering, classification, regression, dimensionality reduction and pre-processing.

Tools: Diffprivlib comes with a number of generic tools for differentially private data analysis. This includes differentially private histograms, following the same format as Numpy's histogram function.

Accountant: The BudgetAccountant class can be used to track privacy budget and calculate total privacy loss using advanced composition techniques.

In [ ]:

```
#The library is designed to run with Python 3. The library can be installed from the PyPi
repository using pip (or pip3)
```

In [4]:

```
!pip install diffprivlib
```

```
Collecting diffprivlib
  Downloading diffprivlib-0.5.1.tar.gz (87 kB)
     |████████████████████████████████| 87 kB 3.3 MB/s
Requirement already satisfied: numpy>=1.19.0 in /usr/local/lib/python3.7/dist-packages (f
rom diffprivlib) (1.21.6)
Requirement already satisfied: setuptools>=49.0.0 in /usr/local/lib/python3.7/dist-packag
es (from diffprivlib) (57.4.0)
Requirement already satisfied: scikit-learn>=0.23.0 in /usr/local/lib/python3.7/dist-pack
ages (from diffprivlib) (1.0.2)
Collecting scipy>=1.5.0
  Downloading scipy-1.7.3-cp37-cp37m-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (38.1
MB)
     |████████████████████████████████| 38.1 MB 1.2 MB/s
Requirement already satisfied: joblib>=0.16.0 in /usr/local/lib/python3.7/dist-packages (
from diffprivlib) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-pack
ages (from scikit-learn>=0.23.0->diffprivlib) (3.1.0)
Building wheels for collected packages: diffprivlib
  Building wheel for diffprivlib (setup.py) ... done
  Created wheel for diffprivlib: filename=diffprivlib-0.5.1-py3-none-any.whl size=162306
sha256=997bb5ab444e7bc0473f2fbbcf2a3bec5590c8304413c7ee4f35251cbd6747d8
  Stored in directory: /root/.cache/pip/wheels/88/fb/35/44ce5d133fbdc88e5cbad820d700b62a4
a7bb7c160408d0493
Successfully built diffprivlib
Installing collected packages: scipy, diffprivlib
  Attempting uninstall: scipy
    Found existing installation: scipy 1.4.1
    Uninstalling scipy-1.4.1:
      Successfully uninstalled scipy-1.4.1
ERROR: pip's dependency resolver does not currently take into account all the packages th
at are installed. This behaviour is the source of the following dependency conflicts.
albumentations 0.1.12 requires imgaug<0.2.7,>=0.2.5, but you have imgaug 0.2.9 which is i
ncompatible.
Successfully installed diffprivlib-0.5.1 scipy-1.7.3
```

In [48]:

```
#We start by importing the required libraries and modules

from sklearn import metrics
from sklearn.cluster import KMeans
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [ ]:

```
#DATSET-IRIS DATASET
#The Iris dataset was used in R.A. Fisher's classic 1936 paper,
#"The Use of Multiple Measurements in Taxonomic Problems" , and can also be found on the
UCI Machine Learning Repository.

#It includes three iris species with 50 samples each as well as some properties about eac
h flower.
#One flower species is linearly separable from the other two, but the other two are not l
inearly separable from each other.

#The columns in this dataset are:

#Id
#SepalLengthCm
#SepalWidthCm
#PetalLengthCm
#PetalWidthCm
#Species
##Please Refer The Refernce Folder In LAB-13 For More info ##
```

In [49]:

```
#Load the dataset- IRIS dataset
iris = pd.read_csv("/content/IRIS.csv")

# iloc()-integer-location based indexing for selection by position.iloc[] is primarily in
teger position based (from 0 to length-1 of the axis), but may also be used with a boolea
n array.
# Ref-" https://www.askpython.com/python/built-in-methods/python-iloc-function"
# First parameter inside the iloc()- rows
# Second Parameter inside the iloc()- column // Randomly Chosen

X =  iris.iloc[:, [0, 1, 2, 3]].values
X
```

Out[49]:

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
       [5. , 3. , 1.6, 0.2],
       [5. , 3.4, 1.6, 0.4],
       [5.2, 3.5, 1.5, 0.2],
       [5.2, 3.4, 1.4, 0.2],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [5.4, 3.4, 1.5, 0.4],
       [5.2, 4.1, 1.5, 0.1],
       [5.5, 4.2, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5. , 3.2, 1.2, 0.2],
       [5.5, 3.5, 1.3, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [4.4, 3. , 1.3, 0.2],
       [5.1, 3.4, 1.5, 0.2],
       [5. , 3.5, 1.3, 0.3],
       [4.5, 2.3, 1.3, 0.3],
       [4.4, 3.2, 1.3, 0.2],
       [5. , 3.5, 1.6, 0.6],
       [5.1, 3.8, 1.9, 0.4],
       [4.8, 3. , 1.4, 0.3],
       [5.1, 3.8, 1.6, 0.2],
       [4.6, 3.2, 1.4, 0.2],
       [5.3, 3.7, 1.5, 0.2],
       [5. , 3.3, 1.4, 0.2],
       [7. , 3.2, 4.7, 1.4],
       [6.4, 3.2, 4.5, 1.5],
       [6.9, 3.1, 4.9, 1.5],
       [5.5, 2.3, 4. , 1.3],
       [6.5, 2.8, 4.6, 1.5],
       [5.7, 2.8, 4.5, 1.3],
       [6.3, 3.3, 4.7, 1.6],
       [4.9, 2.4, 3.3, 1. ],
       [6.6, 2.9, 4.6, 1.3],
       [5.2, 2.7, 3.9, 1.4],
       [5. , 2. , 3.5, 1. ],
       [5.9, 3. , 4.2, 1.5],
       [6. , 2.2, 4. , 1. ],
       [6.1, 2.9, 4.7, 1.4],
       [5.6, 2.9, 3.6, 1.3],
       [6.7, 3.1, 4.4, 1.4],
       [5.6, 3. , 4.5, 1.5],
       [5.8, 2.7, 4.1, 1. ],
       [6.2, 2.2, 4.5, 1.5],
       [5.6, 2.5, 3.9, 1.1],
       [5.9, 3.2, 4.8, 1.8],
       [6.1, 2.8, 4. , 1.3],
       [6.3, 2.5, 4.9, 1.5],
       [6.1, 2.8, 4.7, 1.2],
       [6.4, 2.9, 4.3, 1.3],
       [6.6, 3. , 4.4, 1.4],
       [6.8, 2.8, 4.8, 1.4],
       [6.7, 3. , 5. , 1.7],
       [6. , 2.9, 4.5, 1.5],
       [5.7, 2.6, 3.5, 1. ],
       [5.5, 2.4, 3.8, 1.1],
       [5.5, 2.4, 3.7, 1. ],
       [5.8, 2.7, 3.9, 1.2],
       [6. , 2.7, 5.1, 1.6],
       [5.4, 3. , 4.5, 1.5],
       [6. , 3.4, 4.5, 1.6],
       [6.7, 3.1, 4.7, 1.5],
       [6.3, 2.3, 4.4, 1.3],
       [5.6, 3. , 4.1, 1.3],
       [5.5, 2.5, 4. , 1.3],
       [5.5, 2.6, 4.4, 1.2],
       [6.1, 3. , 4.6, 1.4],
       [5.8, 2.6, 4. , 1.2],
       [5. , 2.3, 3.3, 1. ],
       [5.6, 2.7, 4.2, 1.3],
       [5.7, 3. , 4.2, 1.2],
       [5.7, 2.9, 4.2, 1.3],
       [6.2, 2.9, 4.3, 1.3],
       [5.1, 2.5, 3. , 1.1],
       [5.7, 2.8, 4.1, 1.3],
       [6.3, 3.3, 6. , 2.5],
       [5.8, 2.7, 5.1, 1.9],
       [7.1, 3. , 5.9, 2.1],
       [6.3, 2.9, 5.6, 1.8],
       [6.5, 3. , 5.8, 2.2],
       [7.6, 3. , 6.6, 2.1],
       [4.9, 2.5, 4.5, 1.7],
       [7.3, 2.9, 6.3, 1.8],
       [6.7, 2.5, 5.8, 1.8],
       [7.2, 3.6, 6.1, 2.5],
       [6.5, 3.2, 5.1, 2. ],
       [6.4, 2.7, 5.3, 1.9],
       [6.8, 3. , 5.5, 2.1],
       [5.7, 2.5, 5. , 2. ],
       [5.8, 2.8, 5.1, 2.4],
       [6.4, 3.2, 5.3, 2.3],
       [6.5, 3. , 5.5, 1.8],
       [7.7, 3.8, 6.7, 2.2],
       [7.7, 2.6, 6.9, 2.3],
       [6. , 2.2, 5. , 1.5],
       [6.9, 3.2, 5.7, 2.3],
       [5.6, 2.8, 4.9, 2. ],
       [7.7, 2.8, 6.7, 2. ],
       [6.3, 2.7, 4.9, 1.8],
       [6.7, 3.3, 5.7, 2.1],
       [7.2, 3.2, 6. , 1.8],
       [6.2, 2.8, 4.8, 1.8],
       [6.1, 3. , 4.9, 1.8],
       [6.4, 2.8, 5.6, 2.1],
       [7.2, 3. , 5.8, 1.6],
       [7.4, 2.8, 6.1, 1.9],
       [7.9, 3.8, 6.4, 2. ],
       [6.4, 2.8, 5.6, 2.2],
       [6.3, 2.8, 5.1, 1.5],
       [6.1, 2.6, 5.6, 1.4],
       [7.7, 3. , 6.1, 2.3],
       [6.3, 3.4, 5.6, 2.4],
       [6.4, 3.1, 5.5, 1.8],
       [6. , 3. , 4.8, 1.8],
       [6.9, 3.1, 5.4, 2.1],
       [6.7, 3.1, 5.6, 2.4],
       [6.9, 3.1, 5.1, 2.3],
       [5.8, 2.7, 5.1, 1.9],
       [6.8, 3.2, 5.9, 2.3],
       [6.7, 3.3, 5.7, 2.5],
       [6.7, 3. , 5.2, 2.3],
       [6.3, 2.5, 5. , 1.9],
       [6.5, 3. , 5.2, 2. ],
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3. , 5.1, 1.8]])
```

In [ ]:

```
#CASE1-EVALUATING KMEANS WITHOUT PRIVACY PRESERVING MECHANISM
```

In [ ]:

```
#KMeans with no privacy
#To begin, let's first train a regular (non-private) KMEANS, and test its accuracy.
```

In [ ]:

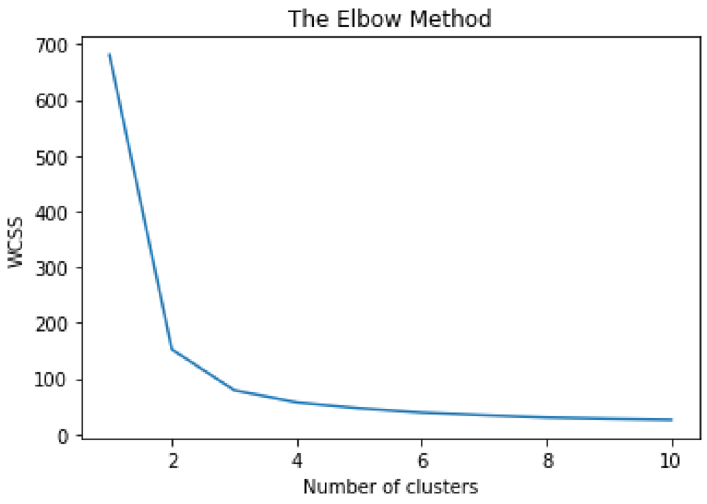
```
#How to Implementing K-Means Clustering ?
#1.Choose the number of clusters k
#2.Select k random points from the data as centroids
#3.Assign all the points to the closest cluster centroid
#4.Recompute the centroids of newly formed clusters
#5.Repeat steps 3 and 4
```

In [57]:

```
#Finding the optimum number of clusters for k-means classification
wcss =[]
for i in range (1,11):
    nonprivate_clf  = KMeans(n_clusters = i, init = 'k-means++', max_iter =300, n_init =
10, random_state = 0)
    nonprivate_clf .fit(X)
    wcss.append( nonprivate_clf.inertia_)
```

In [56]:

```
#Using the elbow method to determine the optimal number of clusters for k-means clusterin
g
# Plot the graph to visualize the Elbow Method to find the optimal number of cluster
plt.plot(range(1,11),wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
##Referred from- https://www.kaggle.com/code/khotijahs1/k-means-clustering-of-iris-datase
t/notebook
```



In [ ]:

```
#Implementing K-Means Clustering
```

In [58]:

```
# Kmeans Ref-"https://scikit-learn.org/stable/modules/clustering.html#k-means"
       # Reference 2 - " https://jakevdp.github.io/PythonDataScienceHandbook/05.11-k-mea
ns.html"


#  KMean Syntax Reference-"https://github.com/scikit-learn/scikit-learn/blob/baf828ca1/sk
```

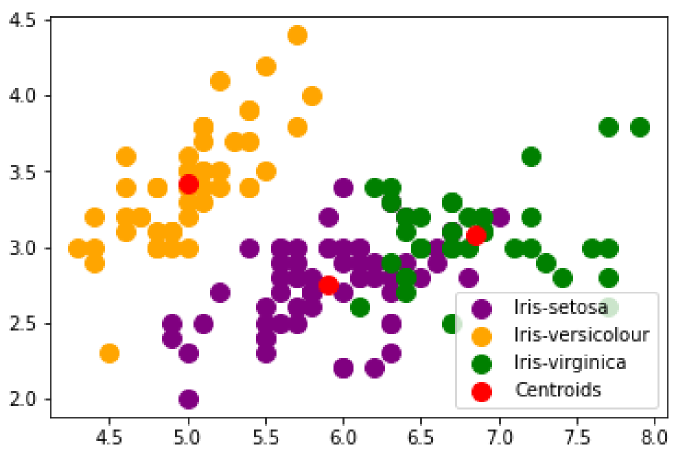
```
learn/cluster/_kmeans.py#L763"
# Applying KMeans to the dataset with the optimal number of cluster
kmeans = KMeans(n_clusters = 3, init = 'k-means++', max_iter = 300, n_init = 10, random_
state = 0)
y_kmeans = kmeans.fit_predict(X)
```

In [59]:

```
#Visualising the clusters
plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 1], s = 100, c = 'purple', label = 'Ir
is-setosa')
plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 1], s = 100, c = 'orange', label = 'Ir
is-versicolour')
plt.scatter(x[y_kmeans == 2, 0], x[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Iri
s-virginica')

#Plotting the centroids of the clusters
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1], s = 100, c = 'r
ed', label = 'Centroids')

plt.legend()
```

Out[59]:

```
<matplotlib.legend.Legend at 0x7f727b5b06d0>
```



In [60]:

```
#Check Accuracy
labels = kmeans.labels_

# check how many of the samples were correctly labeled

correct_labels = sum(y_kmeans == labels)

print("Result: %d out of %d samples were correctly labeled." % (correct_labels, y_kmeans.
size))

print('Accuracy score: {0:0.2f}'. format(correct_labels/float(y_kmeans.size)))
```

```
Result: 150 out of 150 samples were correctly labeled.
Accuracy score: 1.00
```

In [ ]:

```
#Differentially private KMEANS
```

In [ ]:

```
#CASE2-EVALUATING PRIVACY PRESERVING  KMEANS WITHOUT EPILSON AND BOUND VALUE
```

In [ ]:

```
#Using the diffprivlib.models.KMeans module of diffprivlib, we can train a KMeans while s
atisfying differential privacy
#Referred from- https://diffprivlib.readthedocs.io/en/latest/modules/models.html#k-means
```

```
#Syntax- classdiffprivlib.models.KMeans(n_clusters=8, *, epsilon=1.0, bounds=None, accoun
tant=None, **unused_args)
#Syntax Source- https://diffprivlib.readthedocs.io/en/latest/_modules/diffprivlib/models/
k_means.html#KMeans

#K-Means clustering with differential privacy.


#Implements the DPLloyd approach presented in [SCL16], leveraging the sklearn.cluster.KMe
ans class for full integration with Scikit Learn.
#PLEASE Refer the "Reference Folder in LAB 13-> Differentially Private K-Means Clustering
" to Know More about DPLloyd approach
```

In [67]:

```
from diffprivlib.models import KMeans
```

In [89]:

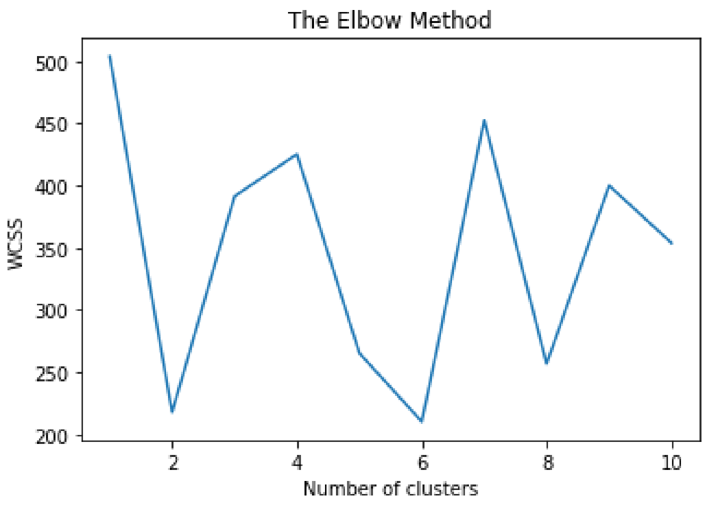
```
#Finding the optimum number of clusters for k-means classification
wcss =[]
for i in range (1,11):
    private_clf  = KMeans()
    private_clf .fit(X)
    wcss.append( private_clf.inertia_)
```

```
/usr/local/lib/python3.7/dist-packages/diffprivlib/models/k_means.py:130: PrivacyLeakWarn
ing: Bounds have not been specified and will be calculated on the data provided.  This wi
ll result in additional privacy leakage. To ensure differential privacy and no additional
privacy leakage, specify `bounds` for each dimension.
  "privacy leakage, specify `bounds` for each dimension.", PrivacyLeakWarning)
/usr/local/lib/python3.7/dist-packages/diffprivlib/models/k_means.py:130: PrivacyLeakWarn
ing: Bounds have not been specified and will be calculated on the data provided.  This wi
ll result in additional privacy leakage. To ensure differential privacy and no additional
privacy leakage, specify `bounds` for each dimension.
  "privacy leakage, specify `bounds` for each dimension.", PrivacyLeakWarning)
/usr/local/lib/python3.7/dist-packages/diffprivlib/models/k_means.py:130: PrivacyLeakWarn
ing: Bounds have not been specified and will be calculated on the data provided.  This wi
ll result in additional privacy leakage. To ensure differential privacy and no additional
privacy leakage, specify `bounds` for each dimension.
  "privacy leakage, specify `bounds` for each dimension.", PrivacyLeakWarning)
/usr/local/lib/python3.7/dist-packages/diffprivlib/models/k_means.py:130: PrivacyLeakWarn
ing: Bounds have not been specified and will be calculated on the data provided.  This wi
ll result in additional privacy leakage. To ensure differential privacy and no additional
privacy leakage, specify `bounds` for each dimension.
  "privacy leakage, specify `bounds` for each dimension.", PrivacyLeakWarning)
/usr/local/lib/python3.7/dist-packages/diffprivlib/models/k_means.py:130: PrivacyLeakWarn
ing: Bounds have not been specified and will be calculated on the data provided.  This wi
ll result in additional privacy leakage. To ensure differential privacy and no additional
privacy leakage, specify `bounds` for each dimension.
  "privacy leakage, specify `bounds` for each dimension.", PrivacyLeakWarning)
/usr/local/lib/python3.7/dist-packages/diffprivlib/models/k_means.py:130: PrivacyLeakWarn
ing: Bounds have not been specified and will be calculated on the data provided.  This wi
ll result in additional privacy leakage. To ensure differential privacy and no additional
privacy leakage, specify `bounds` for each dimension.
  "privacy leakage, specify `bounds` for each dimension.", PrivacyLeakWarning)
/usr/local/lib/python3.7/dist-packages/diffprivlib/models/k_means.py:130: PrivacyLeakWarn
ing: Bounds have not been specified and will be calculated on the data provided.  This wi
ll result in additional privacy leakage. To ensure differential privacy and no additional
privacy leakage, specify `bounds` for each dimension.
  "privacy leakage, specify `bounds` for each dimension.", PrivacyLeakWarning)
/usr/local/lib/python3.7/dist-packages/diffprivlib/models/k_means.py:130: PrivacyLeakWarn
ing: Bounds have not been specified and will be calculated on the data provided.  This wi
ll result in additional privacy leakage. To ensure differential privacy and no additional
privacy leakage, specify `bounds` for each dimension.
  "privacy leakage, specify `bounds` for each dimension.", PrivacyLeakWarning)
/usr/local/lib/python3.7/dist-packages/diffprivlib/models/k_means.py:130: PrivacyLeakWarn
ing: Bounds have not been specified and will be calculated on the data provided.  This wi
ll result in additional privacy leakage. To ensure differential privacy and no additional
privacy leakage, specify `bounds` for each dimension.
  "privacy leakage, specify `bounds` for each dimension.", PrivacyLeakWarning)
```

```
/usr/local/lib/python3.7/dist-packages/diffprivlib/models/k_means.py:130: PrivacyLeakWarn
ing: Bounds have not been specified and will be calculated on the data provided.  This wi
ll result in additional privacy leakage. To ensure differential privacy and no additional
privacy leakage, specify `bounds` for each dimension.
  "privacy leakage, specify `bounds` for each dimension.", PrivacyLeakWarning)
```

In [92]:

```
#Using the elbow method to determine the optimal number of clusters for k-means clusterin
g
# Plot the graph to visualize the Elbow Method to find the optimal number of cluster
plt.plot(range(1,11),wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
##Referred from- https://www.kaggle.com/code/khotijahs1/k-means-clustering-of-iris-datase
t/notebook
```



Since The elbow point in the above graph is difficult to find- We Can Use -Kneedle algorithm Referred From- **https://towardsdatascience.com/detecting-knee-elbow-points-in-a-graph-d13fc517a63c**

In [94]:

```
!pip install --upgrade kneed
```

```
Collecting kneed
  Downloading kneed-0.7.0-py2.py3-none-any.whl (9.4 kB)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from knee
d) (1.7.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (from
kneed) (3.2.2)
Requirement already satisfied: numpy>=1.14.2 in /usr/local/lib/python3.7/dist-packages (f
rom kneed) (1.21.6)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-pack
ages (from matplotlib->kneed) (2.8.2)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib
/python3.7/dist-packages (from matplotlib->kneed) (3.0.8)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-package
s (from matplotlib->kneed) (1.4.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (fr
om matplotlib->kneed) (0.11.0)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-package
s (from kiwisolver>=1.0.1->matplotlib->kneed) (4.1.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from p
ython-dateutil>=2.1->matplotlib->kneed) (1.15.0)
Installing collected packages: kneed
Successfully installed kneed-0.7.0
```

In [99]:

```
# pip install kneed or conda install kneed
from kneed import KneeLocator, DataGenerator
```

In [ ]:

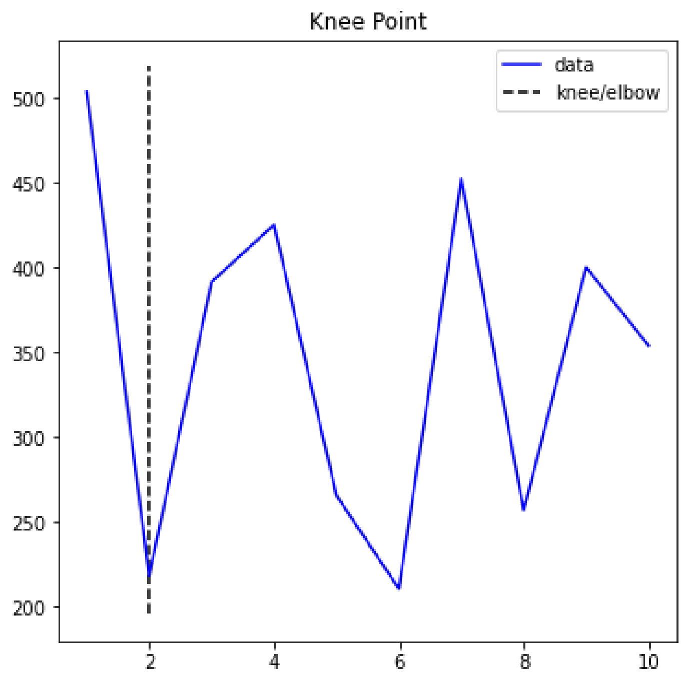
```

```

The knee point is loosely defined as the point of maximum curvature in a system. Identifying this location can be useful in many instances, but in machine learning it can be used for assiting with selection of an appropriate value of k in K-means clustering. Python package, kneed, that can be used to detect the knee or elbow point by attempting an implmentation of the Kneedle algorithm. Referred From- https://www.kaggle.com/code/kevinarvai/knee-elbow-point-detection/notebook

In [111]:

```
#The knee (or elbow) point is calculated simply by instantiating the KneeLocator class wi
th x, y and the appropriate curve and direction.
kneedle = KneeLocator(range(1,11),wcss, S=1.0, curve='convex', direction='decreasing')
# Raw data and knee
kneedle.plot_knee()
# kneedle.elbow store the point of maximum curvature.
print(round(kneedle.elbow, 3))
#Referred From- " https://github.com/arvkevi/kneed"
```

```
2
```



Here only k=2 as we have not specified any parameters inside the KMeans Function.

In [69]:

```
#Implement K Means
clf = KMeans()
clf.fit(X)
```

```
/usr/local/lib/python3.7/dist-packages/diffprivlib/models/k_means.py:130: PrivacyLeakWarn
ing: Bounds have not been specified and will be calculated on the data provided.  This wi
ll result in additional privacy leakage. To ensure differential privacy and no additional
privacy leakage, specify `bounds` for each dimension.
  "privacy leakage, specify `bounds` for each dimension.", PrivacyLeakWarning)
```

Out[69]:

```
KMeans(accountant=BudgetAccountant(spent_budget=[(1.0, 0), (1.0, 0), (1.0, 0), (1.0, 0),
(1.0, 0), ...]),
       bounds=(array([4.3, 2. , 1. , 0.1]), array([7.9, 4.4, 6.9, 2.5])))
```

diffprivlib.models.KMeans can be run without any parameters, although this will throw a warning (we need to specify the bounds parameter to avoid this). The privacy level is controlled by the parameter epsilon, which is

passed to the classifier at initialisation (e.g. KMeans(epsilon=0.1)). The default is epsilon = 1.0. Referred From - "https://github.com/IBM/differential-privacy-library"

In [ ]:

```

```

In [ ]:

```
#Every time the model is trained with .fit(), a different model is produced due to the ra
ndomness of differential privacy.
#Referred From- " https://github.com/IBM/differential-privacy-library"
```

In [112]:

```
kmeans = KMeans(n_clusters = 2, epsilon=1.0, bounds=None)
y_kmeans = kmeans.fit_predict(X)
```

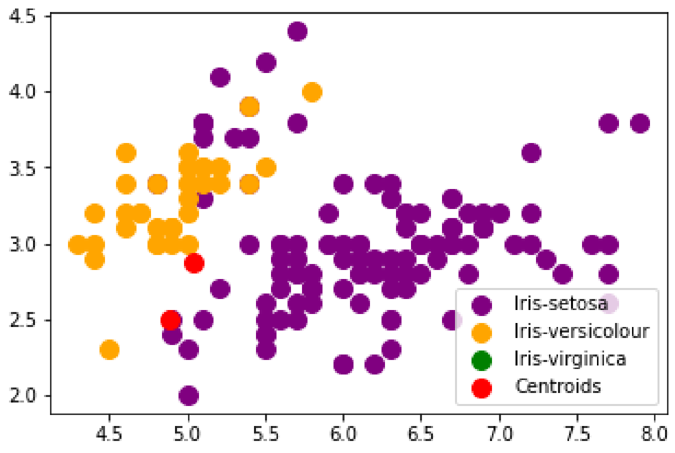
```
/usr/local/lib/python3.7/dist-packages/diffprivlib/models/k_means.py:130: PrivacyLeakWarn
ing: Bounds have not been specified and will be calculated on the data provided.  This wi
ll result in additional privacy leakage. To ensure differential privacy and no additional
privacy leakage, specify `bounds` for each dimension.
  "privacy leakage, specify `bounds` for each dimension.", PrivacyLeakWarning)
```

In [113]:

```
#Visualising the clusters
plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 1], s = 100, c = 'purple', label = 'Ir
is-setosa')
plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 1], s = 100, c = 'orange', label = 'Ir
is-versicolour')
#plt.scatter(x[y_kmeans == 2, 0], x[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Iri
s-virginica')

#Plotting the centroids of the clusters
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1], s = 100, c = 'r
ed', label = 'Centroids')

plt.legend()
```

Out[113]:

```
<matplotlib.legend.Legend at 0x7f727ae61910>
```



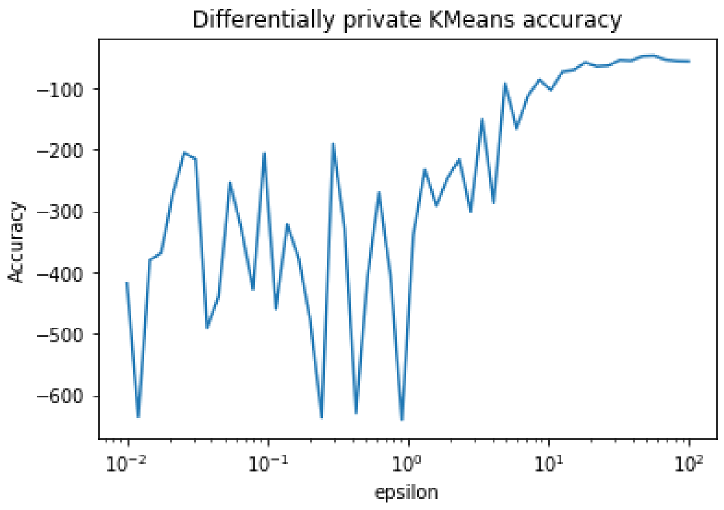
In [ ]:

```
#CASE3-EVALUATING PRIVACY PRESERVING  KMEANS WITH EPILSON AND BOUND VALUE SET
 #evaluate the accuracy of the model for various epsilon values and plot it with matplotl
ib.
```

In [114]:

```
import numpy as np
import matplotlib.pyplot as plt

epsilons = np.logspace(-2, 2, 50)
```

```
bounds = ([4.3, 2.0, 1.1, 0.1], [7.9, 4.4, 6.9, 2.5])
accuracy = list()

for epsilon in epsilons:
    clf = KMeans(bounds=bounds, epsilon=epsilon)
    clf.fit(X)

    accuracy.append(clf.score(X))

plt.semilogx(epsilons, accuracy)
plt.title("Differentially private KMeans accuracy")
plt.xlabel("epsilon")
plt.ylabel("Accuracy")
plt.show()
```


Differentially private KMeans accuracy
Accuracy
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
epsilon