DECISION TREE CLASSIFIER USING GINI INDEX AS IMPURITY MEASURE

1. Load the German Credit card dataset

```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving German Credit Data.csv to German Credit Data.csv
```

2. Create a Pandas Frame for this file and explore its content

```
import pandas as pd
df = pd.read_csv("/content/German Credit Data.csv")
df.shape
df.info()
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   checkin_acc       1000 non-null   object
 1   duration          1000 non-null   int64 
 2   credit_history    1000 non-null   object
 3   amount            1000 non-null   int64 
 4   savings_acc       1000 non-null   object
 5   present_emp_since 1000 non-null   object
 6   inst_rate         1000 non-null   int64 
 7   personal_status   1000 non-null   object
 8   residing_since    1000 non-null   int64 
 9   age               1000 non-null   int64 
 10  inst_plans        1000 non-null   object
 11  num_credits       1000 non-null   int64 
 12  job               1000 non-null   object
 13  status            1000 non-null   int64 
dtypes: int64(7), object(7)
memory usage: 109.5+ KB
```

	checkin_acc	duration	credit_history	amount	savings_acc	present_emp_since	inst_r
0	A11	6	A34	1169	A65	A75	
1	A12	48	A32	5951	A61	A73	
2	A14	12	A34	2096	A61	A74	
3	A11	42	A32	7882	A61	A74	
4	A11	24	A33	4870	A61	A73	

3. Print the first five records and first 7 columns

```
df.iloc[0:5,0:6]
```

	checkin_acc	duration	credit_history	amount	savings_acc	present_emp_since
0	A11	6	A34	1169	A65	A75
1	A12	48	A32	5951	A61	A73
2	A14	12	A34	2096	A61	A74
3	A11	42	A32	7882	A61	A74
4	A11	24	A33	4870	A61	A73

4. Print the first five records and remaining columns

```
df.iloc[0:5,6:14]
```

	inst_rate	personal_status	residing_since	age	inst_plans	num_credits	job	statu
0	4	A93	4	67	A143	2	A173	
1	2	A92	2	22	A143	1	A173	
2	2	A93	3	49	A143	1	A172	
3	2	A93	4	45	A143	1	A173	
4	3	A93	4	53	A143	2	A173	

5. Few of the columns are categorical and are infered as objects. Ex: checkin_acc. Print all unique values of this column

```
df['checkin_acc'].unique()
```

```
array(['A11', 'A12', 'A14', 'A13'], dtype=object)
```

```
df['credit_history'].unique()
```

```
array(['A34', 'A32', 'A33', 'A30', 'A31'], dtype=object)
```

6. Encode all categorical features using one-hot encoding/dummy encoding. A feature with n values is encoded using (n-1) values, retaining the first one (drop_first = True)

```
X_features = list(df.columns)
X_features.remove('status')
encoded_df = pd.get_dummies(df[X_features],drop_first=True)
print(list(encoded_df.columns))
```

```
['duration', 'amount', 'inst_rate', 'residing_since', 'age', 'num_credits', 'checkin_acc_A12', 'checkin_acc_A13', 'checkin_acc_A14', 'cr
```

7. Make independent features of the encoded frame as X and column 'status' as dependent feature.

```
X = encoded_df
Y = df['status']
```

8. Divide data into 70% training and 30% as testing.

```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.3, random_state = 42)
```

9. Train a decision tree model using Gini INdex and depth of 3

```
from sklearn.tree import DecisionTreeClassifier
clf= DecisionTreeClassifier(criterion = 'gini',max_depth=3)
clf.fit(X_train,Y_train)
```

```
DecisionTreeClassifier(max_depth=3)
```

10. Make predictions on test/validation data

```
pred_Y = clf.predict(X_test)
```

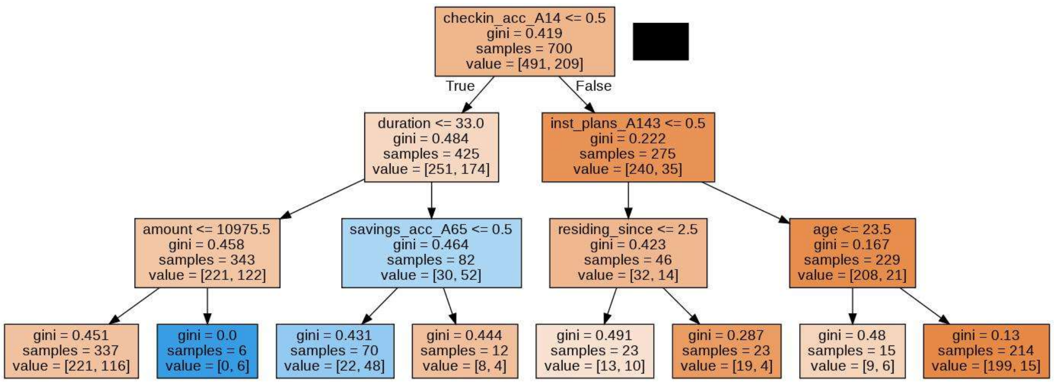
11. Print the confusion matrix, accuracy and AUC score of this model on test set

```
from sklearn import metrics
print("confusion Matrix is \n",metrics.confusion_matrix(pred_Y,Y_test))
```

```
print("Accuracy is",metrics.accuracy_score(pred_Y,Y_test))
print("AUC Score is", metrics.roc_auc_score(pred_Y,Y_test))
     confusion Matrix is
      [[198  71]
      [ 11  20]]
     Accuracy is 0.7266666666666667
     AUC Score is 0.6906103849382419
```

12. Visualize the tree using grapghviz and pydotplus libraries

```
from sklearn.tree import export_graphviz
import pydotplus as pdot
from IPython.display import Image
export_graphviz(clf,out_file = "tree.odt",feature_names = X_train.columns,filled = True)
graph = pdot.graphviz.graph_from_dot_file("tree.odt")
graph.write_jpg("tree.png")
Image(filename = "tree.png")
```



Hyperparameter Tuning for kNN for Predicting Heart Disease

1. Import "heart.csv".

```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving heart.csv to heart.csv
```

2. Import Library

```
import pandas as pd
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV
```

3. Load the Dataset into a frame

```
df = pd.read_csv('/content/heart.csv')
```

4. Print the description, dimensions and first five records of the frame.

```
print(df.info())
print(df.shape)
print(df.head())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       303 non-null    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trestbps  303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalach   303 non-null    int64  
 8   exang     303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64  
 11  ca        303 non-null    int64  
 12  thal      303 non-null    int64  
 13  target    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
None
(303, 14)
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   63    1   3       145   233    1        0      150      0      2.3      0   
1   37    1   2       130   250    0        1      187      0      3.5      0   
2   41    0   1       130   204    0        0      172      0      1.4      2   
3   56    1   1       120   236    0        1      178      0      0.8      2   
4   57    0   0       120   354    0        1      163      1      0.6      2   

   ca  thal  target  
0   0     1       1  
1   0     2       1  
2   0     2       1  
3   0     2       1  
4   0     2       1  
```

5. Check whether the data has any missing value in any column.

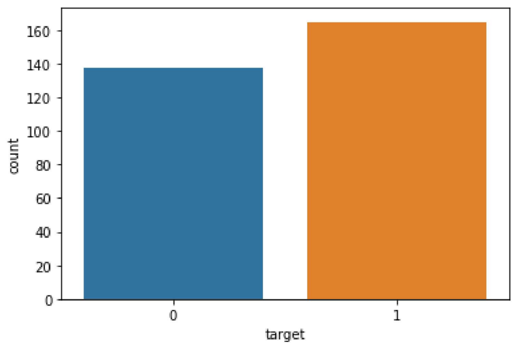
```
df.isnull().sum()
```

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

6. Check whether the data has balanced class distribution. Class target = 0 indicates "Heart Disease" and target = 1 indicates "No Heart Disease".

```
sns.countplot(df['target'])
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWarning: Pass th
  warnings.warn(
<matplotlib.axes._subplots.AxesSubplot at 0x7f45d541e340>
```



7. Create input features X, target Y, classifier object, train-test-split using 80-20% split

```
x = df.drop(columns=['target'])
y = df['target']
knn = KNeighborsClassifier()
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=4)
```

8. Train model

```
#Training the model
knn.fit(x_train, y_train)
```

```
KNeighborsClassifier()
```

9. Validate model on test set

```
y_pred = knn.predict(x_test)
```

10. Print Classification Report on test data

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support
```

```
              0       0.48      0.52      0.50        25
              1       0.65      0.61      0.63        36

       accuracy                           0.57        61
      macro avg       0.56      0.57      0.56        61
   weighted avg       0.58      0.57      0.58        61
```

11. Print AUC score on test data

```
roc_auc_score(y_test, y_pred)
```

```
0.5655555555555556
```

The performance of the model is vey poor. Hence hyperparameters of kNN to be tuned using GridSearchCV.

12. Hyperparameter tuning using GridSearchCV. Set the parameters a)leaf-size= 1 to 15, b)n_neighbors = 1 to 10 and c) distance metric, p = 1, 2. When p =1 its Manhattan and p = 2 its Euclidean distance. GridSearchCV uses CV to search for the optimal values of the hyperparameters. It accepts the hyperparameters as a dictionary.

```
leaf_size = list(range(1,15))
n_neighbors = list(range(1,10))
p=[1,2]
hyperparameters = dict(leaf_size=leaf_size, n_neighbors=n_neighbors, p=p)
```

13. Train a new kNN model using GridSearchCV.

```
knn_2 = KNeighborsClassifier()
clf = GridSearchCV(knn_2, hyperparameters, cv=10, scoring = 'roc_auc')
best_model = clf.fit(x,y)
```

14. Print the best values of the hyperparameters.

```
#Nilai hyperpaameters terbaik
print('Best leaf_size:', best_model.best_estimator_.get_params()['leaf_size'])
print('Best p:', best_model.best_estimator_.get_params()['p'])
print('Best n_neighbors:', best_model.best_estimator_.get_params()['n_neighbors'])
print('Best Score:', best_model.best_score_)
```

```
Best leaf_size: 9
Best p: 1
Best n_neighbors: 7
Best Score: 0.7483536683904332
```

15. Validate the model on test data

```
y_pred = best_model.predict(x_test)
```

16. Print classification report and AUC score of the model on test data

```
print(classification_report(y_test, y_pred))
print("AUC SCORE is",roc_auc_score(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.72      0.72      0.72        25
           1       0.81      0.81      0.81        36

    accuracy                           0.77        61
   macro avg       0.76      0.76      0.76        61
weighted avg       0.77      0.77      0.77        61

AUC SCORE is 0.7627777777777778
```

1. Implement the "user-based similarity" method of Collaborative Filtering. Upload the "ratings.csv". Create a frame for this file.

```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving rating.csv to rating.csv

```
import pandas as pd
df = pd.read_csv("/content/rating.csv")
df.head(5)
```

	userId	movieId	rating	timestamp
0	1	2	3.5	02-04-2005 23:53
1	1	29	3.5	02-04-2005 23:31
2	1	32	3.5	02-04-2005 23:33
3	1	47	3.5	02-04-2005 23:32
4	1	50	3.5	02-04-2005 23:29

2. Drop the timestamp column and find the number of unique movies and unique users.

```
df.drop('timestamp', axis = 1, inplace = True)
print("No. of unique users =",len(df['userId'].unique()))
print("No. of unique movies =",len(df['movieId'].unique()))
```

```
No. of unique users = 7120
No. of unique movies = 14026
```

3. Create a pivot table or matrix with users as rows and movies as columns. Matrix entries will represent the ratings given by the users. This will be a sparse matrix and those movies not watched will be NaN. Replace all such NaN values by zeros.

```
user_movies_df = df.pivot(index = "userId", columns = "movieId", values = "rating").reset_index(drop = True)
user_movies_df.index = df.userId.unique()
user_movies_df.fillna(0, inplace = True)
```

4. Compute the cosine similarity matrix between all pairs of users.The diagonal values of this matrix will be 1. Print the matrix.

```
from sklearn.metrics import pairwise_distances
from scipy.spatial.distance import cosine, correlation
user_sim = 1 - pairwise_distances(user_movies_df.values, metric = "cosine")
#covert this matrix to a frame
user_sim_df = pd.DataFrame(user_sim)
#set the index and column names to user ids
user_sim_df.index = df.userId.unique()
user_sim_df.columns = df.userId.unique()
user_sim_df.iloc[0:5, 0:5]
```

	1	2	3	4	5
1	1.000000	0.102916	0.261198	0.029091	0.139199
2	0.102916	1.000000	0.180124	0.064014	0.140042
3	0.261198	0.180124	1.000000	0.057323	0.156755
4	0.029091	0.064014	0.057323	1.000000	0.300828
5	0.139199	0.140042	0.156755	0.300828	1.000000

5. Set the diagonal values as 0.0, since we need to find other users who are similar to a specific user.

```
import numpy as np
np.fill_diagonal(user_sim, 0)
user_sim_df.iloc[0:5, 0:5]
```

	1	2	3	4	5
1	0.000000	0.102916	0.261198	0.029091	0.139199
2	0.102916	0.000000	0.180124	0.064014	0.140042
3	0.261198	0.180124	0.000000	0.057323	0.156755
4	0.029091	0.064014	0.057323	0.000000	0.300828
5	0.139199	0.140042	0.156755	0.300828	0.000000

6. Filter similar users. To find most similar users, the maximum values of each column can be filtered. For example, the most similar user to first 5 users with userId from 1 to 5 can be printed as

```
user_sim_df.idxmax(axis=1)[0:5]
```

```
1    2595
2    5964
3     475
4     242
5    1367
dtype: int64
```

7. This result says that the most similar user to user1 is userId 2595 and so on.This also means that both have watched several movies in common and rated very similarly. This can be verified with the movies dataset. Load the movies dataset, drop the "genres" column.

```
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving movie.csv to movie.csv

```
movies_df = pd.read_csv("/content/movie.csv")
movies_df.drop('genres', axis = 1, inplace = True)
movies_df.iloc[0:5, 0:5]
```

	movieId	title
0	1	Toy Story (1995)
1	2	Jumanji (1995)
2	3	Grumpier Old Men (1995)
3	4	Waiting to Exhale (1995)
4	5	Father of the Bride Part II (1995)

8. Find common movies of similar users.

```
def get_user_similar_movies(user1, user2):
  #Inner join of movies watched by both users will give the common movies
  common = df[df.userId == user1].merge(df[df.userId == user2], on = "movieId", how = "inner")
  #merge this result with movie details
  return common.merge(movies_df, on = "movieId")
```

9. Find the common movies of user 1 and user 2595.Print only those common movies with ratings above 4.

```
common_movies = get_user_similar_movies(1,2595 )
#print(common_movies)
common_movies[(common_movies.rating_x >=4.0) & ((common_movies.rating_y >=4.0))]
```

	userId_x	movieId	rating_x	userId_y	rating_y	title
3	1	260	4.0	2595	5.0	Star Wars: Episode IV - A New Hope (1977)
4	1	293	4.0	2595	4.0	Léon: The Professional (a.k.a. The Professiona...
5	1	296	4.0	2595	5.0	Pulp Fiction (1994)
6	1	318	4.0	2595	5.0	Shawshank Redemption, The (1994)
7	1	541	4.0	2595	4.5	Blade Runner (1982)
12	1	1036	4.0	2595	4.5	Die Hard (1988)
13	1	1097	4.0	2595	5.0	E.T. the Extra-Terrestrial (1982)
16	1	1196	4.5	2595	4.5	Star Wars: Episode V - The Empire Strikes Back...
17	1	1198	4.5	2595	5.0	Raiders of the Lost Ark (Indiana Jones and the...
18	1	1200	4.0	2595	4.5	Aliens (1986)
20	1	1214	4.0	2595	5.0	Alien (1979)
21	1	1215	4.0	2595	4.0	Army of Darkness (1993)
22	1	1219	4.0	2595	5.0	Psycho (1960)
24	1	1240	4.0	2595	4.5	Terminator, The (1984)
26	1	1259	4.0	2595	4.5	Stand by Me (1986)
29	1	1278	4.0	2595	4.5	Young Frankenstein (1974)
31	1	1333	4.0	2595	4.5	Birds, The (1963)
33	1	1387	4.0	2595	5.0	Jaws (1975)
36	1	2118	4.0	2595	4.0	Dead Zone, The (1983)
38	1	2288	4.0	2595	4.5	Thing, The (1982)
43	1	2762	4.0	2595	4.5	Sixth Sense, The (1999)
47	1	3153	4.0	2595	4.5	7th Voyage of Sinbad, The (1958)
48	1	3499	4.0	2595	4.0	Misery (1990)
52	1	4306	4.0	2595	4.5	Shrek (2001)
56	1	4993	5.0	2595	4.0	Lord of the Rings: The Fellowship of the Ring,...
58	1	5952	5.0	2595	4.5	Lord of the Rings: The Two Towers, The (2002)
63	1	6774	4.0	2595	4.0	Videodrome (1983)
65	1	7153	5.0	2595	5.0	Lord of the Rings: The Return of the King, The...
67	1	7757	4.0	2595	4.0	Jason and the Argonauts (1963)
69	1	8507	5.0	2595	4.0	Freaks (1932)
70	1	8636	4.5	2595	4.0	Spider-Man 2 (2004)
71	1	8961	4.0	2595	4.5	Incredibles, The (2004)

10. Finding user similarity does not work for new users. We need to wait until the new user buys a few items and rates them. Only then user preferences can be founnd and recommendations can be made. This is called the "cold-start" problem in recommender systems. This is overcome by using item-based similarity methods.

Implement the k_Means Clustering using "Income.csv"

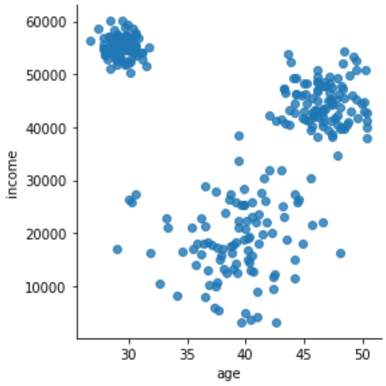
```
from google.colab import files
uploaded = files.upload()
```

Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving Income Data.csv to Income Data (1).csv

1. Create a data frame and visualize the natural groupings in the dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sn
df = pd.read_csv("/content/Income Data.csv")
sn.lmplot("age", "income", data = df, fit_reg = False, size = 4)
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the fo
  FutureWarning
/usr/local/lib/python3.6/dist-packages/seaborn/regression.py:580: UserWarning: The `size` pa
  warnings.warn(msg, UserWarning)
<seaborn.axisgrid.FacetGrid at 0x7f8a0e1b7fd0>
```



2. The above groupings are mostly segmented using income, since it has a huge range. Scale of age is 0 to 60 and income is from 0 to 50000. Hence Euclidean distance will always be dominated by income and not age. Hence all features need to be normalised to a uniform scale before clustering.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_df = scaler.fit_transform(df[["age", "income"]])
scaled_df[0:5]
```

```
array([[ 1.3701637 ,  0.09718548],
       [-1.3791283 ,  0.90602749],
       [ 1.10388844,  0.51405021],
       [ 0.23849387, -1.27162408],
       [-0.35396857, -1.32762083]])
```

3. PLotting customers with their segments

```
from sklearn.cluster import KMeans
clusters = KMeans(3)
clusters.fit(scaled_df)
df["clusterid"] = clusters.labels_
markers = ['+', '^', '*']
sn.lmplot("age", "income", data = df, hue = "clusterid", fit_reg = False, markers  = markers, size = 4)
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the fo
  FutureWarning
/usr/local/lib/python3.6/dist-packages/seaborn/regression.py:580: UserWarning: The `size` pa
  warnings.warn(msg, UserWarning)
<seaborn.axisgrid.FacetGrid at 0x7f8a0dcf8780>
```

4. Print the cluster centers using the original dataframe. Cluster centres explain the characteristics of the cluster and helps us to interpret the clusters. Print the cluster centres to understand the average age and income of each cluster.

```
clusters = KMeans(3)
clusters.fit(df)
df["new_clusterid"] = clusters.labels_
df.groupby("new_clusterid")['age', 'income'].agg(["mean", 'std']).reset_index()
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: FutureWarning: Indexing with
  after removing the cwd from sys.path.
```

	new_clusterid	age		income	
		mean	std	mean	std
0	0	39.174479	3.626068	18144.791667	6745.241906
1	1	31.700435	6.122122	54675.652174	2362.224320
2	2	46.419101	2.289620	43053.932584	3613.769632

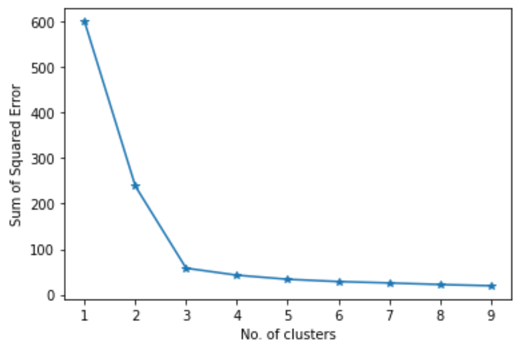
Double-click (or enter) to edit

5. So Cluster 0 has a mean age of 39 and income of 18K. Low age and low income. CLuster 1 has a mean age of 37 and income of 54K. Mid age and high income. CLuster 2 has a mean age of 46 and income of 43K. High age and medium income. The actual age and income of a customer within a cluster will vary from the cluster centers and is called the cluster variance. This is given by WCSS - within cluster sum of squares.

6. Find the optimum number of clusters that may exist using Elbow Method. Try with number of clusters from 1 to 10. In each case print the total variance using "inertia" parameter of the clusters.

```
cluster_range = range(1,10)
cluster_errors = []
for num_clusters in cluster_range:
  clusters = KMeans(num_clusters)
  clusters.fit(scaled_df)
  cluster_errors.append(clusters.inertia_)
plt.figure(figsize = (6,4))
plt.plot(cluster_range, cluster_errors, marker = "*")
plt.xlabel("No. of clusters")
plt.ylabel("Sum of Squared Error")
```

```
Text(0, 0.5, 'Sum of Squared Error')
```



Double-click (or enter) to edit

7. The figure indicates the elbow point is 3, this means there might exist three clusters in the data set.

1. Implement the "user-based similarity" method of Collaborative Filtering. Upload the "ratings.csv". Create a frame for this file.

```
from google.colab import files
uploaded = files.upload()
```

Choose files No file chosen Cancel upload

```
import pandas as pd
df = pd.read_csv("/content/rating.csv")
df.head(5)
```

	userId	movieId	rating	timestamp
0	1	2	3.5	02-04-2005 23:53
1	1	29	3.5	02-04-2005 23:31
2	1	32	3.5	02-04-2005 23:33
3	1	47	3.5	02-04-2005 23:32
4	1	50	3.5	02-04-2005 23:29

2. Drop the timestamp column and find the number of unique movies and unique users.

```
df.drop('timestamp', axis = 1, inplace = True)
print("No. of unique users =",len(df['userId'].unique()))
print("No. of unique movies =",len(df['movieId'].unique()))
```

```
No. of unique users = 7120
No. of unique movies = 14026
```

3. Create a pivot table or matrix with users as rows and movies as columns. Matrix entries will represent the ratings given by the users. This will be a sparse matrix and those movies not watched will be NaN. Replace all such NaN values by zeros.

```
user_movies_df = df.pivot(index = "userId", columns = "movieId", values = "rating").reset_index(drop = True)
user_movies_df.index = df.userId.unique()
user_movies_df.fillna(0, inplace = True)
```

4. Compute the cosine similarity matrix between all pairs of users.The diagonal values of this matrix will be 1. Print the matrix.

```
from sklearn.metrics import pairwise_distances
from scipy.spatial.distance import cosine, correlation
user_sim = 1 - pairwise_distances(user_movies_df.values, metric = "cosine")
#covert this matrix to a frame
user_sim_df = pd.DataFrame(user_sim)
#set the index and column names to user ids
user_sim_df.index = df.userId.unique()
user_sim_df.columns = df.userId.unique()
user_sim_df.iloc[0:5, 0:5]
```

	1	2	3	4	5
1	1.000000	0.102916	0.261198	0.029091	0.139199
2	0.102916	1.000000	0.180124	0.064014	0.140042
3	0.261198	0.180124	1.000000	0.057323	0.156755
4	0.029091	0.064014	0.057323	1.000000	0.300828
5	0.139199	0.140042	0.156755	0.300828	1.000000

5. Set the diagonal values as 0.0, since we need to find other users who are similar to a specific user.

```
import numpy as np
np.fill_diagonal(user_sim, 0)
user_sim_df.iloc[0:5, 0:5]
```

	1	2	3	4	5
1	0.000000	0.102916	0.261198	0.029091	0.139199
2	0.102916	0.000000	0.180124	0.064014	0.140042

6. Filter similar users. To find most similar users, the maximum values of each column can be filtered. For example, the most similar user to first 5 users with userId from 1 to 5 can be printed as

```
user_sim_df.idxmax(axis=1)[0:5]
```

```
1    2595
2    5964
3     475
4     242
5    1367
dtype: int64
```

7. This result says that the most similar user to user1 is userId 2595 and so on.This also means that both have watched several movies in common and rated very similarly. This can be verified with the movies dataset. Load the movies dataset, drop the "genres" column.

```
uploaded = files.upload()
```

Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving movie.csv to movie.csv
```

```
movies_df = pd.read_csv("/content/movie.csv")
movies_df.drop('genres', axis = 1, inplace = True)
movies_df.iloc[0:5, 0:5]
```

	movieId	title
0	1	Toy Story (1995)
1	2	Jumanji (1995)
2	3	Grumpier Old Men (1995)
3	4	Waiting to Exhale (1995)
4	5	Father of the Bride Part II (1995)

8. Find common movies of similar users.

```
def get_user_similar_movies(user1, user2):
  #Inner join of movies watched by both users will give the common movies
  common = df[df.userId == user1].merge(df[df.userId == user2], on = "movieId", how = "inner")
  #merge this result with movie details
  return common.merge(movies_df, on = "movieId")
```

9. Find the common movies of user 1 and user 2595.Print only those common movies with ratings above 4.

```
common_movies = get_user_similar_movies(1,2595 )
#print(common_movies)
common_movies[(common_movies.rating_x >=4.0) & ((common_movies.rating_y >=4.0))]
```

	userId_x	movieId	rating_x	userId_y	rating_y	title
3	1	260	4.0	2595	5.0	Star Wars: Episode IV - A New Hope (1977)
4	1	293	4.0	2595	4.0	Léon: The Professional (a.k.a. The Professiona...
5	1	296	4.0	2595	5.0	Pulp Fiction (1994)
6	1	318	4.0	2595	5.0	Shawshank Redemption, The (1994)
7	1	541	4.0	2595	4.5	Blade Runner (1982)
12	1	1036	4.0	2595	4.5	Die Hard (1988)
13	1	1097	4.0	2595	5.0	E.T. the Extra-Terrestrial (1982)
16	1	1196	4.5	2595	4.5	Star Wars: Episode V - The Empire Strikes Back...
17	1	1198	4.5	2595	5.0	Raiders of the Lost Ark (Indiana Jones and the...
18	1	1200	4.0	2595	4.5	Aliens (1986)
20	1	1214	4.0	2595	5.0	Alien (1979)
21	1	1215	4.0	2595	4.0	Army of Darkness (1993)
22	1	1219	4.0	2595	5.0	Psycho (1960)
24	1	1240	4.0	2595	4.5	Terminator, The (1984)
26	1	1259	4.0	2595	4.5	Stand by Me (1986)
29	1	1278	4.0	2595	4.5	Young Frankenstein (1974)

10. Finding user similarity does not work for new users. We need to wait until the new user buys a few items and rates them. Only then user preferences can be founnd and recommendations can be made. This is called the "cold-start" problem in recommender systems. This is overcome by using item-based similarity methods.

38	1	2288	4.0	2595	4.5	Thing, The (1982)
43	1	2762	4.0	2595	4.5	Sixth Sense, The (1999)
47	1	3153	4.0	2595	4.5	7th Voyage of Sinbad, The (1958)
48	1	3499	4.0	2595	4.0	Misery (1990)
52	1	4306	4.0	2595	4.5	Shrek (2001)
56	1	4993	5.0	2595	4.0	Lord of the Rings: The Fellowship of the Ring,...
58	1	5952	5.0	2595	4.5	Lord of the Rings: The Two Towers, The (2002)
63	1	6774	4.0	2595	4.0	Videodrome (1983)
65	1	7153	5.0	2595	5.0	Lord of the Rings: The Return of the King, The...
67	1	7757	4.0	2595	4.0	Jason and the Argonauts (1963)