```
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q https://www-us.apache.org/dist/spark/spark-2.4.4/spark-2.4.4-bin-hadoop2.7.tgz
!tar xf spark-2.4.4-bin-hadoop2.7.tgz
!pip install -q findspark
```

```
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-2.4.4-bin-hadoop2.7"
```

```
!pip install pyspark
```

```
Requirement already satisfied: pyspark in /usr/local/lib/python2.7/dist-packages (2.4.4)
Requirement already satisfied: py4j==0.10.7 in /usr/local/lib/python2.7/dist-packages (from pyspark) (0.10.7)
```

```
from itertools import chain
import numpy as np
import pandas as pd
```

```
rating = pd.read_csv("ratings.dat.txt", sep = "::", header = None )
```

```
/usr/local/lib/python2.7/dist-packages/ipykernel_launcher.py:1: ParserWarning: Falling back to the 'python' engine be
  """Entry point for launching an IPython kernel.
```

```
movies = pd.read_csv("movies.dat.txt", sep = "::", header = None )
```

```
/usr/local/lib/python2.7/dist-packages/ipykernel_launcher.py:1: ParserWarning: Falling back to the 'python' engine be
  """Entry point for launching an IPython kernel.
```

```
df = pd.DataFrame(movies)
df.columns = ['MovieID','Title','Genres']
```

```
df.head()
```

```
df.head()
```

	MovieID	Title	Genres
0	1	Toy Story (1995)	Animation|Children's|Comedy
1	2	Jumanji (1995)	Adventure|Children's|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama
4	5	Father of the Bride Part II (1995)	Comedy

```
def chainer(s):
    return list(chain.from_iterable(s.str.split('|')))

lens = df['Genres'].str.split('|').map(len)

result = pd.DataFrame({'MovieID': np.repeat(df['MovieID'], lens),
                    'Title': np.repeat(df['Title'], lens),
                    'Genres': chainer(df['Genres'])})

final_result = result

final_result["Values"] = 1

final_result.head()
```

Genres	MovieID	Title	Values
0 Children's	1	Toy Story (1995)	1

```
pivot_table = pd.pivot_table(final_result, index = ["MovieID","Title"],columns = "Genres", values= 'Values')
pivot_table.fillna(0,inplace=True)
```

```
pivot_table.head()
```

MovieID	Title \ Genres	Action	Adventure	Animation	Children's	Comedy	Crime	Documentary	Drama	Fantasy	Film-Noir	Horro
1	Toy Story (1995)	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.
2	Jumanji (1995)	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.
3	Grumpier Old Men (1995)	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.
4	Waiting to Exhale (1995)	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.
5	Father of the Bride Part II (1995)	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.

```
pivot_table.to_csv("final_value_cluster_1",header= False, index=False)
```

```
pivot_table.to_csv("final_Pivot_cluster.csv")
```

K-Means Clustering

```
from pyspark.mllib.clustering import KMeans, KMeansModel
```

```
from pyspark.mllib.clustering import KMeans, KMeansModel
import pyspark
from pyspark import SparkContext


from numpy import array
from math import sqrt


from sklearn.metrics import mean_squared_error


sc = SparkContext()


sample = sc.textFile("final_value_cluster_1")




parsedData_2= sample.map(lambda line: array([float(x) for x in line.split(',')]))


def error(point):
    center = clusters_2.centers[clusters_2.predict(point)]
    return sqrt(sum([x**2 for x in (point - center)]))


for i in range(2,20):
  clusters_2 = KMeans.train(parsedData_2, i, maxIterations=100)
  WSSSE = parsedData_2.map(lambda point: error(point)).reduce(lambda x, y: x + y)
  print("Within Root Mean Squared Error for {} cluster= ".format(i) + str(WSSSE))
```

```
Within Root Mean Squared Error for 2 cluster= 3535.30068609
Within Root Mean Squared Error for 3 cluster= 3089.96709514
Within Root Mean Squared Error for 4 cluster= 3292.16465079
Within Root Mean Squared Error for 5 cluster= 3147.29156554
Within Root Mean Squared Error for 6 cluster= 2759.88888426
Within Root Mean Squared Error for 7 cluster= 2650.70544163
Within Root Mean Squared Error for 8 cluster= 2506.26767664
Within Root Mean Squared Error for 9 cluster= 2616.00316378
Within Root Mean Squared Error for 10 cluster= 2493.65492451
Within Root Mean Squared Error for 11 cluster= 2236.99873751
Within Root Mean Squared Error for 12 cluster= 2007.31866628
Within Root Mean Squared Error for 13 cluster= 2135.11940395
Within Root Mean Squared Error for 14 cluster= 2189.35649247
Within Root Mean Squared Error for 15 cluster= 2199.99382364
Within Root Mean Squared Error for 16 cluster= 1953.07398947
Within Root Mean Squared Error for 17 cluster= 2156.81081775
Within Root Mean Squared Error for 18 cluster= 2327.21823079
Within Root Mean Squared Error for 19 cluster= 1764.07461817
```

```
clusters_2 = KMeans.train(parsedData_2, 18, maxIterations=100)
WSSSE = parsedData_2.map(lambda point: error(point)).reduce(lambda x, y: x + y)
print("Within Root Mean Squared Error for 18 cluster= " + str(WSSSE))
```

```
Within Root Mean Squared Error for 18 cluster= 1836.1814153
```

```
labels = clusters_2.predict(parsedData_2)
```

```
labels_value = labels.collect()
```

```
max(labels_value)
```

```
17
```

```
len(labels_value)
```

```
3883
```

```
data_set = pd.read_csv("final_Pivot_cluster.csv")

data_set['cluster_label'] = labels_value

data_set.head()
```

.me	Documentary	Drama	Fantasy	Film-Noir	Horror	Musical	Mystery	Romance	Sci-Fi	Thriller	War	Western	cluster_label
).0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4
).0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4
).0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	3
).0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	11
).0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5

```
data_Movie_Cluster = data_set[['MovieID','cluster_label']]

data_Movie_Cluster.head()
```

	MovieID	cluster_label
0	1	4
1	2	4
2	3	3
3	4	11
4	5	5

```
data = {}

for i in range(len(data_Movie_Cluster)):
  data[data_Movie_Cluster.iloc[i,0]] = data_Movie_Cluster.iloc[i,1]
```

```
print(len(data))
```

```
3883
```

```
print(data)
```

```
{1: 4, 2: 4, 3: 3, 4: 11, 5: 5, 6: 14, 7: 3, 8: 4, 9: 9, 10: 8, 11: 3, 12: 0, 13: 4, 14: 10, 15: 13, 16: 14, 17: 1, 1
```

```
dataRating = pd.read_csv("ratings.dat.txt", sep="::",header=None)
```

```
/usr/local/lib/python2.7/dist-packages/ipykernel_launcher.py:1: ParserWarning: Falling back to the 'python' engine be
  """Entry point for launching an IPython kernel.
```

```
dataRating.head()
```

	0	1	2	3
0	1	1193	5	978300760
1	1	661	3	978302109
2	1	914	3	978301968
3	1	3408	4	978300275
4	1	2355	5	978824291

```
dataRating.columns = ['UserID','MovieID','Ratings','Time']

data_frame = pd.merge(dataRating,data_Movie_Cluster, how = 'inner', on = 'MovieID' )

data_frame.sort_values('UserID',ascending= True, inplace= True)

data_frame.head()
```

	UserID	MovieID	Ratings	Time	cluster_label
0	1	1193	5	978300760	10
28501	1	48	5	978824351	4
13819	1	938	4	978301752	0
51327	1	1207	4	978300719	10
31152	1	1721	4	978300055	1

```
dataFrame_2 = data_frame.groupby(['UserID','cluster_label'])['Ratings'].mean()

dataFrame_2.head()
```

```
UserID  cluster_label
1       0                4.500000
        1                3.333333
        2                4.000000
        3                3.000000
        4                4.166667
Name: Ratings, dtype: float64
```

```
Final_data_frame = pd.DataFrame(dataFrame_2)
```

```
Final_data_frame.head()
```

		Ratings
UserID	**cluster_label**	
1	**0**	4.500000
	1	3.333333
	2	4.000000
	3	3.000000
	4	4.166667

```
Pivot_data = pd.pivot_table(Final_data_frame,index='UserID', columns='cluster_label',values='Ratings')
```

```
Pivot_data.fillna(0,inplace= True)
```

```
Pivot_data.head()
```

cluster_label	0	1	2	3	4	5	6	7	8	9	10
UserID											
1	4.500	3.333333	4.000000	3.000000	4.166667	4.000000	0.000000	4.000000	4.000000	4.666667	4.428571
2	3.000	3.800000	3.000000	3.857143	0.000000	3.000000	0.000000	4.000000	3.666667	3.571429	4.000000
3	5.000	0.000000	0.000000	3.500000	4.000000	3.833333	0.000000	0.000000	4.166667	3.800000	4.000000
4	5.000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	4.000000	3.428571	4.666667	5.000000
5	2.875	3.294118	3.454545	2.714286	4.000000	3.571429	3.666667	2.333333	2.428571	2.846154	3.054545

```
Predicted_value = []

for i in range(len(data_frame)):
  Predicted_value.append(Pivot_data.iloc[data_frame.iloc[i,0]-1,data_frame.iloc[i,4]])

 print(len(Predicted_value))
```

```
1000209
```

```
data_frame['Predicted_values'] = Predicted_value
```

```
data_frame.head()
```

UserID MovieID Ratings Time cluster_label Predicted_values

```
RMSE = np.sqrt(mean_squared_error(data_frame['Ratings'], data_frame['Predicted_values']))

print("The Root Mean Squared Error rate is {}".format(RMSE))
```

```
The Root Mean Squared Error rate is 0.956297732126
```

51327 1 1207 4 978300719 10 4.428571

ALS Code

```
import math

from pyspark.mllib.recommendation import ALS, Rating, MatrixFactorizationModel

rating_data = sc.textFile("ratings.dat.txt")

rating_dataset = rating_data.map(lambda x: x.split("::")).cache()

data_set_rating =rating_dataset.map(lambda x: Rating(int(x[0]),int(x[1]),float(x[2]))).cache()
print data_set_rating.take(3)
```

```
[Rating(user=1, product=1193, rating=5.0), Rating(user=1, product=661, rating=3.0), Rating(user=1, product=914, ratin
```

```
training_RDD, validation_RDD, test_RDD = data_set_rating.randomSplit([6, 2, 2], seed=0L)
validation_for_predict_RDD = validation_RDD.map(lambda x: (x[0], x[1]))
test_for_predict_RDD = test_RDD.map(lambda x: (x[0], x[1]))

seed = 5L
iterations = [10,20,30]
regularization_parameter = [0.1,0.01]
ranks = [2,4,6,8,10,12]
errors = []
err = 0
tolerance = 0.02
```

```
min_error = float('inf')
best_rank = -1
best_iteration = -1
best_regularization = -1
for rank in ranks:
  for iterat in iterations:
    for regular in regularization_parameter:

      model = ALS.train(training_RDD, rank, seed=seed, iterations=iterat,
                        lambda_=regular)
      predictions = model.predictAll(validation_for_predict_RDD).map(lambda r: ((r[0], r[1]), r[2]))
      rates_and_preds = validation_RDD.map(lambda r: ((int(r[0]), int(r[1])), float(r[2]))).join(predictions)
      error = math.sqrt(rates_and_preds.map(lambda r: (r[1][0] - r[1][1])**2).mean())
      errors.append(error)
      err += 1
      print('For rank %s, Iteration %s and Regular Parameter %s the RMSE is %s' % (rank,iterat, regular, error))
      if error < min_error:
        min_error = error
        best_rank = rank
        best_iteration = iterat
        best_regularization = regular


print('The best model was trained with rank %s, iterations %s, regularized paramater %s' % (best_rank, best_iteration
```

```
For rank 2, Iteration 10 and Regular Parameter 0.1 the RMSE is 0.893315032091
For rank 2, Iteration 10 and Regular Parameter 0.01 the RMSE is 0.889336594875
For rank 2, Iteration 20 and Regular Parameter 0.1 the RMSE is 0.892140942022
For rank 2, Iteration 20 and Regular Parameter 0.01 the RMSE is 0.889089340889
For rank 2, Iteration 30 and Regular Parameter 0.1 the RMSE is 0.891807209162
For rank 2, Iteration 30 and Regular Parameter 0.01 the RMSE is 0.88908514981
For rank 4, Iteration 10 and Regular Parameter 0.1 the RMSE is 0.886260195446
For rank 4, Iteration 10 and Regular Parameter 0.01 the RMSE is 0.886787869352
For rank 4, Iteration 20 and Regular Parameter 0.1 the RMSE is 0.879370515147
For rank 4, Iteration 20 and Regular Parameter 0.01 the RMSE is 0.887318771069
For rank 4, Iteration 30 and Regular Parameter 0.1 the RMSE is 0.877967645533
For rank 4, Iteration 30 and Regular Parameter 0.01 the RMSE is 0.886955392245
For rank 6, Iteration 10 and Regular Parameter 0.1 the RMSE is 0.876189172989
For rank 6, Iteration 10 and Regular Parameter 0.01 the RMSE is 0.89623016961
For rank 6, Iteration 20 and Regular Parameter 0.1 the RMSE is 0.871393876257
For rank 6, Iteration 20 and Regular Parameter 0.01 the RMSE is 0.895555048902
For rank 6, Iteration 30 and Regular Parameter 0.1 the RMSE is 0.87040252925
For rank 6, Iteration 30 and Regular Parameter 0.01 the RMSE is 0.895796914978
For rank 8, Iteration 10 and Regular Parameter 0.1 the RMSE is 0.876701840863
For rank 8, Iteration 10 and Regular Parameter 0.01 the RMSE is 0.914981748012
For rank 8, Iteration 20 and Regular Parameter 0.1 the RMSE is 0.869385600323
For rank 8, Iteration 20 and Regular Parameter 0.01 the RMSE is 0.911816635773
For rank 8, Iteration 30 and Regular Parameter 0.1 the RMSE is 0.86750501064
For rank 8, Iteration 30 and Regular Parameter 0.01 the RMSE is 0.910336669883
For rank 10, Iteration 10 and Regular Parameter 0.1 the RMSE is 0.875259664709
For rank 10, Iteration 10 and Regular Parameter 0.01 the RMSE is 0.926934441681
For rank 10, Iteration 20 and Regular Parameter 0.1 the RMSE is 0.867337057463
For rank 10, Iteration 20 and Regular Parameter 0.01 the RMSE is 0.928930205935
For rank 10, Iteration 30 and Regular Parameter 0.1 the RMSE is 0.865641671038
For rank 10, Iteration 30 and Regular Parameter 0.01 the RMSE is 0.930189150663
For rank 12, Iteration 10 and Regular Parameter 0.1 the RMSE is 0.872532683651
For rank 12, Iteration 10 and Regular Parameter 0.01 the RMSE is 0.944298297152
For rank 12, Iteration 20 and Regular Parameter 0.1 the RMSE is 0.866354878552
For rank 12, Iteration 20 and Regular Parameter 0.01 the RMSE is 0.943283244587
For rank 12, Iteration 30 and Regular Parameter 0.1 the RMSE is 0.86499812804
For rank 12, Iteration 30 and Regular Parameter 0.01 the RMSE is 0.943854663496
The best model was trained with rank 12, iterations 30, regularized paramater 0.1
```