In [1]:

```
from pyspark.sql import functions as F
from pyspark.sql import DataFrameNaFunctions as DFna
from pyspark.sql.functions import udf, col, when
import matplotlib.pyplot as plt
import pyspark as ps
import os, sys, requests, json


spark = ps.sql.SparkSession.builder \
            .master("local[4]") \
            .appName("building recommender") \
            .getOrCreate() # create a spark session

sc = spark.sparkContext # create a spark context
```

In [2]:

```
# read movies CSV
movies_df = spark.read.csv(r"C:\Users\hp\Anaconda3\ml-20m\movies.csv",
                           header=True,       # use headers or not
                           quote='"',         # char for quotes
                           sep=",",           # char for separation
                           inferSchema=True)  # do we infer schema or not ?
movies_df.printSchema()
```

```
root
 |-- movieId: integer (nullable = true)
 |-- title: string (nullable = true)
 |-- genres: string (nullable = true)
```

In [3]:

```
print("line count: {}".format(movies_df.count()))
```

```
line count: 27278
```

In [4]:

```
# read ratings CSV
ratings_df = spark.read.csv(r"C:\Users\hp\Anaconda3\ml-20m\movies.csv",
                           header=True,       # use headers or not
                           quote='"',         # char for quotes
                           sep=",",           # char for separation
                           inferSchema=True)  # do we infer schema or not ?
ratings_df.printSchema()
```

```
root
 |-- movieId: integer (nullable = true)
 |-- title: string (nullable = true)
 |-- genres: string (nullable = true)
```

In [6]:

```
print("line count: {}".format(movies_df.count()))
```

```
line count: 27278
```

In [7]:

```
# read ratings CSV
ratings_df = spark.read.csv(r"C:\Users\hp\Anaconda3\ml-20m\ratings.csv",
                           header=True,       # use headers or not
                           quote='"',         # char for quotes
```

```
                          quote='"',          # char for quotes
                          sep=",",            # char for separation
                          inferSchema=True)   # do we infer schema or not ?
ratings_df.printSchema()
```

```
root
 |-- userId: integer (nullable = true)
 |-- movieId: integer (nullable = true)
 |-- rating: double (nullable = true)
 |-- timestamp: integer (nullable = true)
```

In [8]:

```
ratings = ratings_df.rdd

numRatings = ratings.count()
numUsers = ratings.map(lambda r: r[0]).distinct().count()
numMovies = ratings.map(lambda r: r[1]).distinct().count()

print ("Got %d ratings from %d users on %d movies." % (numRatings, numUsers, numMovies))
```

```
Got 1048575 ratings from 7120 users on 14026 movies.
```

In [9]:

```
movies_counts = ratings_df.groupBy(col("movieId")).agg(F.count(col("rating")).alias("counts"))
movies_counts.show()
```

```
+-------+------+
|movieId|counts|
+-------+------+
|   3997|   113|
|   1580|  1917|
|   3918|    59|
|   2366|   314|
|   3175|   735|
|   4519|   102|
|   1591|   280|
|    471|   587|
|  36525|    58|
|  44022|   113|
|   2866|    93|
|   1645|   636|
|   5803|    49|
|  54190|    77|
|   1088|   601|
|    833|    72|
|   8638|   193|
|    463|    16|
|   1959|   264|
|   2659|    12|
+-------+------+
only showing top 20 rows
```

In [10]:

```
ratings_df.take(5)
```

Out[10]:

```
[Row(userId=1, movieId=2, rating=3.5, timestamp=1112486027),
 Row(userId=1, movieId=29, rating=3.5, timestamp=1112484676),
 Row(userId=1, movieId=32, rating=3.5, timestamp=1112484819),
 Row(userId=1, movieId=47, rating=3.5, timestamp=1112484727),
 Row(userId=1, movieId=50, rating=3.5, timestamp=1112484580)]
```

In [11]:

```
movies_df.take(5)
```

Out[11]:

```
[Row(movieId=1, title='Toy Story (1995)', genres='Adventure|Animation|Children|Comedy|Fantasy'),
 Row(movieId=2, title='Jumanji (1995)', genres='Adventure|Children|Fantasy'),
 Row(movieId=3, title='Grumpier Old Men (1995)', genres='Comedy|Romance'),
 Row(movieId=4, title='Waiting to Exhale (1995)', genres='Comedy|Drama|Romance'),
 Row(movieId=5, title='Father of the Bride Part II (1995)', genres='Comedy')]
```

In [12]:

```
training_df, validation_df, test_df = ratings_df.randomSplit([.6, .2, .2], seed=0)
training_df
```

Out[12]:

```
DataFrame[userId: int, movieId: int, rating: double, timestamp: int]
```

In [13]:

```
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml import Pipeline
from pyspark.sql import Row
import numpy as np
import math
```

In [14]:

```
seed = 5
iterations = 10
regularization_parameter = 0.1
ranks = range(4, 12)
errors = []
err = 0
tolerance = 0.02
```

In [15]:

```
min_error = float('inf')
best_rank = -1
best_iteration = -1

for rank in ranks:
    als = ALS(maxIter=iterations, regParam=regularization_parameter, rank=rank, userCol="userId", i
temCol="movieId", ratingCol="rating")
    model = als.fit(training_df)
    predictions = model.transform(validation_df)
    new_predictions = predictions.filter(col('prediction') != np.nan)
    evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating", predictionCol="prediction
")
    rmse = evaluator.evaluate(new_predictions)
    errors.append(rmse)

    print ("For rank %s the RMSE is %s" % (rank, rmse))
    if rmse < min_error:
        min_error = rmse
        best_rank = rank
print ("The best model was trained with rank %s" % best_rank)
```

```
For rank 4 the RMSE is 0.8424761576591667
For rank 5 the RMSE is 0.8409045595932452
For rank 6 the RMSE is 0.8390556846852776
For rank 7 the RMSE is 0.8358031159893976
For rank 8 the RMSE is 0.8361783043159049
For rank 9 the RMSE is 0.8357399502399812
For rank 10 the RMSE is 0.8360879795861683
For rank 11 the RMSE is 0.8363140732959184
The best model was trained with rank 9
```

In [16]:

```
als = ALS(maxIter=iterations, regParam=regularization_parameter, rank=rank, userCol="userId", itemC
ol="movieId", ratingCol="rating")
paramGrid = ParamGridBuilder() \
    .addGrid(als.regParam, [0.1, 0.01]) \
    .addGrid(als.rank, range(4, 12)) \
    .build()
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating", predictionCol="prediction")
crossval = CrossValidator(estimator=als,
                          estimatorParamMaps=paramGrid,
                          evaluator=evaluator,
                          numFolds=5)
cvModel = crossval.fit(training_df)
```

In [17]:

```
cvModel_pred = cvModel.transform(validation_df)
cvModel_pred = cvModel_pred.filter(col('prediction') != np.nan)
rmse = evaluator.evaluate(cvModel_pred)
print ("the rmse for optimal grid parameters with cross validation is: {}".format(rmse))
```

```
the rmse for optimal grid parameters with cross validation is: 0.8424761576591668
```

In [18]:

```
final_als = ALS(maxIter=10, regParam=0.1, rank=6, userCol="userId", itemCol="movieId", ratingCol="r
ating")
final_model = final_als.fit(training_df)
final_pred = final_model.transform(validation_df)
final_pred = final_pred.filter(col('prediction') != np.nan)
rmse = evaluator.evaluate(final_pred)
print ("the rmse for optimal grid parameters with cross validation is: {}".format(rmse))
```

```
the rmse for optimal grid parameters with cross validation is: 0.8390556846852776
```

In [19]:

```
# read links CSV
links_df = spark.read.csv(r'C:\Users\hp\Anaconda3\data\movies\links.csv',
                          header=True,       # use headers or not
                          quote='"',         # char for quotes
                          sep=",",           # char for separation
                          inferSchema=True)  # do we infer schema or not ?
links_df.printSchema()
```

```
root
 |-- movieId: integer (nullable = true)
 |-- imdbId: integer (nullable = true)
 |-- tmdbId: integer (nullable = true)
```

In [20]:

```
np.random.seed(42)
user_id = np.random.choice(numUsers)
```

In [24]:

```
new_user_ratings = ratings_df.filter(ratings_df.userId == user_id)
new_user_ratings.sort('rating', ascending=True).take(10) # top rated movies for this user
```

Out[24]:

```
[Row(userId=860, movieId=1226, rating=1.0, timestamp=979799492),
 Row(userId=860, movieId=3594, rating=1.0, timestamp=979797774),
 Row(userId=860, movieId=1277, rating=1.0, timestamp=979798379),
 Row(userId=860, movieId=339, rating=1.0, timestamp=979801171),
 Row(userId=860, movieId=1441, rating=1.0, timestamp=979797722),
 Row(userId=860, movieId=282, rating=1.0, timestamp=979799950),
```

```
 Row(userId=860, movieId=1694, rating=1.0, timestamp=979799783),
 Row(userId=860, movieId=597, rating=1.0, timestamp=979800445),
 Row(userId=860, movieId=1704, rating=1.0, timestamp=979799811),
 Row(userId=860, movieId=2468, rating=1.0, timestamp=979800214)]
```

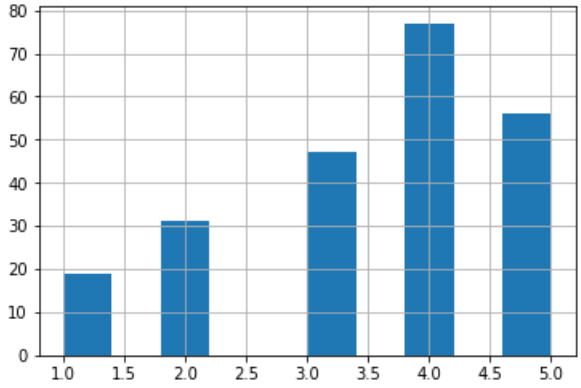
In [22]:

```
new_user_ratings.describe('rating').show()
```

```
+-------+------------------+
|summary|            rating|
+-------+------------------+
|  count|               230|
|   mean|3.5217391304347827|
| stddev|1.2281119537530796|
|    min|               1.0|
|    max|               5.0|
+-------+------------------+
```

In [25]:

```
new_user_ratings.toPandas()['rating'].hist()
plt.show()
```




In [26]:

```
new_user_rated_movieIds = [i.movieId for i in new_user_ratings.select('movieId').distinct().collect
()]
movieIds = [i.movieId for i in movies_counts.filter(movies_counts.counts > 25).select('movieId').di
stinct().collect()]
new_user_unrated_movieIds = list(set(movieIds) - set(new_user_rated_movieIds))
```

In [27]:

```
import time
num_ratings = len(new_user_unrated_movieIds)
cols = ('userId', 'movieId', 'timestamp')
timestamps = [int(time.time())] * num_ratings
userIds = [user_id] * num_ratings
# ratings = [0] * num_ratings
new_user_preds = spark.createDataFrame(zip(userIds, new_user_unrated_movieIds, timestamps), cols)
```

In [28]:

```
new_user_preds = final_model.transform(new_user_preds).filter(col('prediction') != np.nan)
```

In [29]:

```
new_user_preds.sort('prediction', ascending=False).take(10)
```

Out[29]:

```
[Row(userId=860, movieId=1076, timestamp=1575594138, prediction=5.186517715454102),
```

```
 Row(userId=860, movieId=8157, timestamp=1575594138, prediction=5.111594200134277),
 Row(userId=860, movieId=2899, timestamp=1575594138, prediction=4.881644248962402),
 Row(userId=860, movieId=4006, timestamp=1575594138, prediction=4.842452049255371),
 Row(userId=860, movieId=42725, timestamp=1575594138, prediction=4.828301429748535),
 Row(userId=860, movieId=59118, timestamp=1575594138, prediction=4.796186447143555),
 Row(userId=860, movieId=214, timestamp=1575594138, prediction=4.68204927444458),
 Row(userId=860, movieId=6857, timestamp=1575594138, prediction=4.638857841491699),
 Row(userId=860, movieId=36276, timestamp=1575594138, prediction=4.621146202087402),
 Row(userId=860, movieId=4105, timestamp=1575594138, prediction=4.6033735275268555)]
```

In [30]:

```
api_key="efcfb5ada3f2766c2e7cefa9522746c4"
headers = {'Accept': 'application/json'}
payload = {'api_key': api_key}
response = requests.get("http://api.themoviedb.org/3/configuration", params=payload, headers=header
s)
response = json.loads(response.text)
base_url = response['images']['base_url'] + 'w185'

def get_poster(tmdb_id, base_url):

    # Query themoviedb.org API for movie poster path.
    movie_url = 'http://api.themoviedb.org/3/movie/{:}/images'.format(tmdb_id)
    headers = {'Accept': 'application/json'}
    payload = {'api_key': api_key}
    response = requests.get(movie_url, params=payload, headers=headers)
    file_path = json.loads(response.text)['posters'][0]['file_path']
    return base_url + file_path
```

In [31]:

```
from IPython.display import Image
from IPython.display import display
```

In [32]:

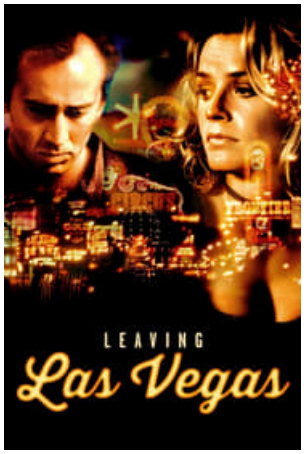
```
new_user_ratings = new_user_ratings.sort('rating', ascending=False).join(links_df, new_user_ratings
.movieId == links_df.movieId)
```

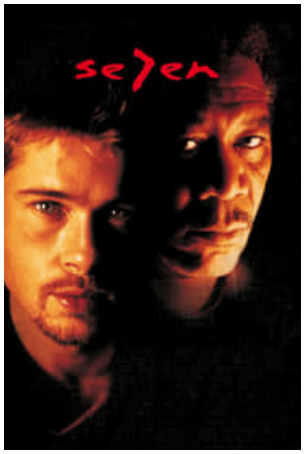
In [33]:

```
posters = tuple(Image(url=get_poster(movie.tmdbId, base_url)) for movie in new_user_ratings.take(10
))
```

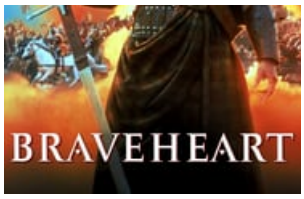
In [34]:

```
display(*posters)
```

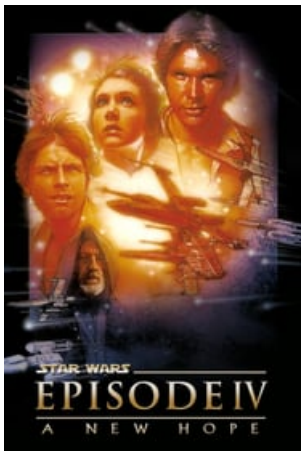


LEAVING
Las Vegas


se7en


the
Usual Suspects


BRAVEHEART


INTERVIEW
WITH THE
VAMPIRE


STAR WARS
EPISODE IV
A NEW HOPE


PULP FICTION
10¢


THE
SHAWSHANK
REDEMPTION

In [35]:

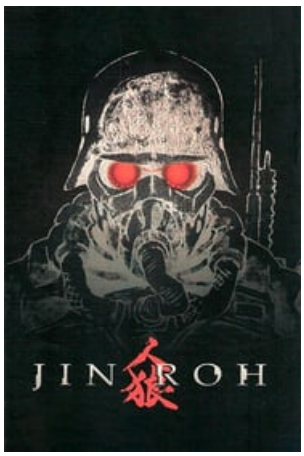
```
new_user_preds = new_user_preds.sort('prediction', ascending=False).join(links_df, new_user_preds.m
ovieId == links_df.movieId)
```

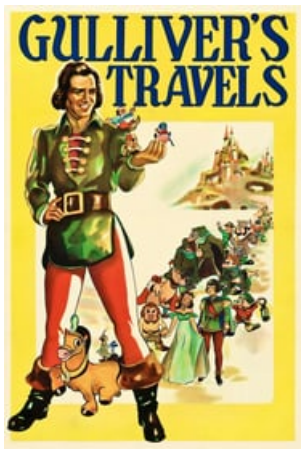
In [36]:

```
posters = tuple(Image(url=get_poster(movie.tmdbId, base_url)) for movie in new_user_preds.take(10))
```

In [37]:

```
display(*posters)
```



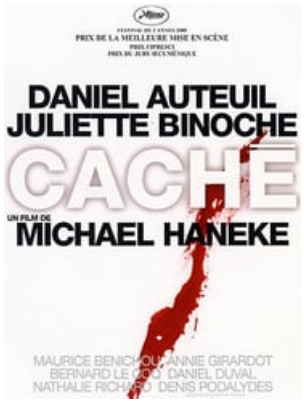
THE TRANSFORMERS
THE MOVIE


GRANDMA'S BOY


una película de mike leigh
happy
un cuento
sobre la felicidad
(Happy-Go-Lucky)


BEFORE THE RAIN


NINJA
SCROLL


DANIEL AUTEUIL
JULIETTE BINOCHE
CACHE
UN FILM DE
MICHAEL HANEKE


THE
EVIL
DEAD

In [ ]: