

综合实验二 电影数据分析

实验基本信息：

时间：

实验类型： ☐验证性 ☐设计性 ☒综合性

班级： 班级代码：

理论老师： 实验指导： 邱开金

实验报告提交说明：

本次实验需要撰写实验报告，实验报告填写时以完成实验任务为目的，先简要回答完成实验任务需要的步骤或需要执行的命令代码，再配以结果截图予以说明，图片数量不宜过多，能说明问题即可。最后配上总结，并提交到教师指定位置。

实验目的：

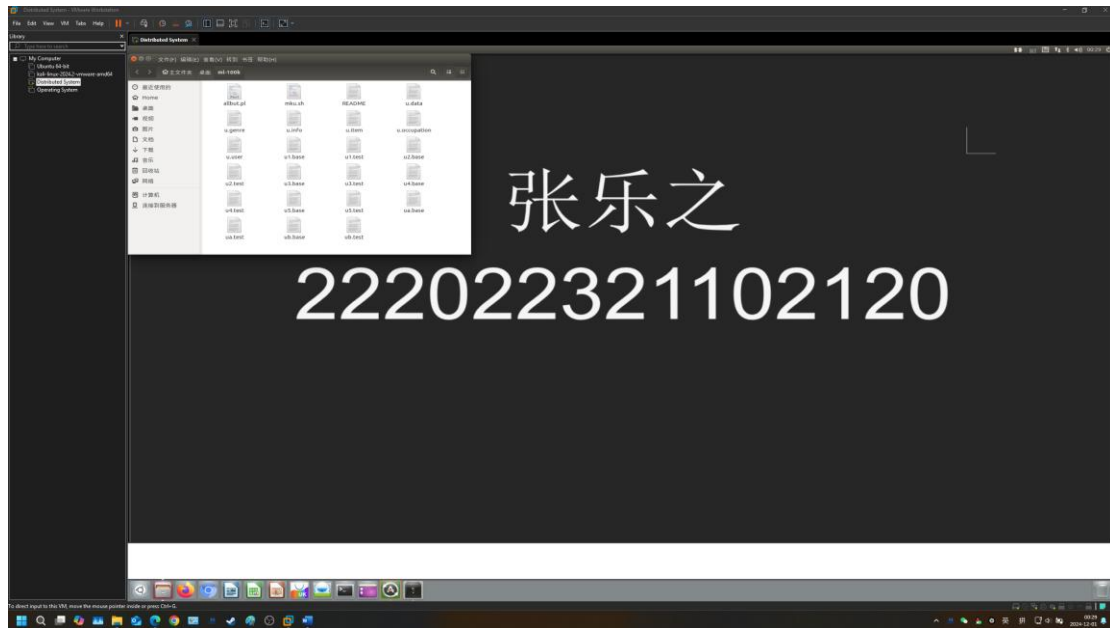
1. 熟悉数据预处理方法
2. 熟练应用 RDD 的算子对数据进行处理分析
3. 熟练使用 DataFrame 的算子对数据进行分析处理
4. 根据情况实际情况完成 RDD 与 DataFrame 的相互转换
5. 掌握阅读 Mlib 官方文档进行机器学习的办法
6. 根据实际任务选择具体的 Mlib 算法进行应用
7. 熟悉数据存储方法
8. 了解数据可视化方法。

实验任务：

使用 Spark MLLib 中的推荐算法接口，实现电影推荐，对使用 Spark 进行推荐有个直观的认识。

- 1、数据集下载 MovieLens 数据集包含多个用户对多部电影的评级数据，也包括电影元数

通过 QQ 群文件下载数据集并导入 Linux 本地解压：



据信息和用户属性信息。本次下载数据 100k 版本。

2、上传到 hdfs 系统（本地也可）

3、读取数据集，进行预处理，并转化为 Row 对象、RDD 对象，最终为 DataFrame 对象。读取用户数据(u.user)转换为 DataFrame,并注册临时表为 user

代码：

```
# 读取用户数据 u.user
```

```
user_schema = ["userId", "age", "gender", "occupation", "zipCode"]
```

```
users = sc.textFile("file:///home/hadoop/桌面/ml-100k/u.user") \
```

```
    .map(lambda line: line.split('|')) \
```

```
    .map(lambda p: Row(userId=int(p[0]), age=int(p[1]), gender=p[2],
```

```
    occupation=p[3], zipCode=p[4])) \
```

```
    .toDF()
```

```
rows = sc.textFile("file:///home/hadoop/桌面/ml-100k/u.user") \
```

```
    .map(lambda line: line.split('|'))
```

```
print("Rows:")
```

```
print(rows.take(5))
```

```
users.createOrReplaceTempView("user")
```

```
users.show()
```

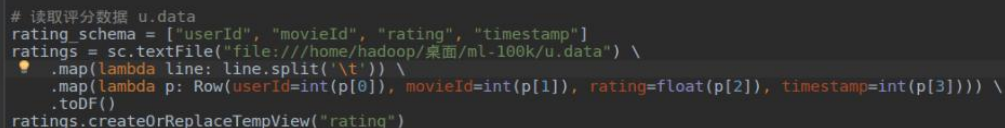
截图：

```
# 读取用户数据 u.user
user_schema = ["userId", "age", "gender", "occupation", "zipCode"]
users = sc.textFile("file:///home/hadoop/桌面/ml-100k/u.user") \
    .map(lambda line: line.split('|')) \
    .map(lambda p: Row(userId=int(p[0]), age=int(p[1]), gender=p[2], occupation=p[3], zipCode=p[4])) \
    .toDF()
rows = sc.textFile("file:///home/hadoop/桌面/ml-100k/u.user") \
    .map(lambda line: line.split('|'))
print("Rows:")
print(rows.take(5))

users.createOrReplaceTempView("user")
users.show()
```

4、读取评分数据(u.data)转换为 DataFrame,并注册临时表为 rating

```
# 读取评分数据 u.data
rating_schema = ["userId", "movieId", "rating", "timestamp"]
ratings = sc.textFile("file:///home/hadoop/桌面/ml-100k/u.data") \
    .map(lambda line: line.split('\t')) \
    .map(lambda p: Row(userId=int(p[0]), movieId=int(p[1]), rating=float(p[2]),
timestamp=int(p[3]))) \
    .toDF()
ratings.createOrReplaceTempView("rating")
```



```
# 读取评分数据 u.data
rating_schema = ["userId", "movieId", "rating", "timestamp"]
ratings = sc.textFile("file:///home/hadoop/桌面/ml-100k/u.data") \
    .map(lambda line: line.split('\t')) \
    .map(lambda p: Row(userId=int(p[0]), movieId=int(p[1]), rating=float(p[2]), timestamp=int(p[3]))) \
    .toDF()
ratings.createOrReplaceTempView("rating")
```

5、读取电影信息(u.item)转换为 DataFrame,并注册临时表为 film

```
item_schema = ["movieId", "title", "releaseDate", "videoReleaseDate",
"IMDbURL"] + \
    ["unknown", "action", "adventure", "animation", "children",
"comedy", "crime", "documentary", "drama",
    "fantasy", "filmNoir", "horror", "musical", "mystery", "romance",
"sciFi", "thriller", "war", "western"]
items = sc.textFile("file:///home/hadoop/桌面/ml-100k/u.item") \
    .map(lambda line: line.split('|')) \
    .map(lambda p: Row(movieId=int(p[0]), title=p[1], releaseDate=p[2],
videoReleaseDate=p[3], IMDbURL=p[4],
        unknown=int(p[5]), action=int(p[6]),
adventure=int(p[7]), animation=int(p[8]),
        children=int(p[9]), comedy=int(p[10]),
crime=int(p[11]), documentary=int(p[12]),
        drama=int(p[13]), fantasy=int(p[14]),
filmNoir=int(p[15]), horror=int(p[16]),
        musical=int(p[17]), mystery=int(p[18]),
romance=int(p[19]), sciFi=int(p[20]),
        thriller=int(p[21]), war=int(p[22]), western=int(p[23]))) \
    .toDF()

items.createOrReplaceTempView("film")
```

```
# 读取电影数据 u.item
item_schema = ["movieId", "title", "releaseDate", "videoReleaseDate", "IMDbURL"] + \
["unknown", "action", "adventure", "animation", "children", "comedy", "crime", "documentary", "drama",
"fantasy", "filmNoir", "horror", "musical", "mystery", "romance", "sciFi", "thriller", "war", "western"]
items = sc.textFile("file:///home/hadoop/桌面/ml-100k/u.item") \
.map(lambda line: line.split('\t')) \
.map(lambda p: Row(movieId=int(p[0]), title=p[1], releaseDate=p[2], videoReleaseDate=p[3], IMDbURL=p[4],
unknown=int(p[5]), action=int(p[6]), adventure=int(p[7]), animation=int(p[8]),
children=int(p[9]), comedy=int(p[10]), crime=int(p[11]), documentary=int(p[12]),
drama=int(p[13]), fantasy=int(p[14]), filmNoir=int(p[15]), horror=int(p[16]),
musical=int(p[17]), mystery=int(p[18]), romance=int(p[19]), sciFi=int(p[20]),
thriller=int(p[21]), war=int(p[22]), western=int(p[23])) \
.toDF()

items.createOrReplaceTempView("film")
```

6、统计分析任务（数据来源还包括 u.occupation, u.genre 等）：

1）电影中的体裁分布（每种体裁对应的电影个数）

```
genre_columns = ["unknown", "action", "adventure", "animation", "children",
"comedy", "crime", "documentary", "drama",
"fantasy", "filmNoir", "horror", "musical", "mystery",
"romance", "sciFi", "thriller", "war", "western"]
```

```
genre_distribution = items.select(
    [(F.sum(F.col(col)).alias(col)) for col in genre_columns]
).toPandas()
```

```
print("Genre Distribution:")
print(genre_distribution)
```

```
genre_columns = ["unknown", "action", "adventure", "animation", "children", "comedy", "crime", "documentary", "drama",
"fantasy", "filmNoir", "horror", "musical", "mystery", "romance", "sciFi", "thriller", "war", "western"]

genre_distribution = items.select(
    [(F.sum(F.col(col)).alias(col)) for col in genre_columns]
).toPandas()

print("Genre Distribution:")
print(genre_distribution)
```



2）以“1 表示年龄<18, 2 表示年龄在[18,24], 3 表示年龄在[25,29], 4 表示年龄在[30,34], 5 表示年龄在[35,39], 6 表示年龄在[40,49], 7 和 8 表示年龄>=50, 0 表示未知”来统计每种年龄阶段对于所有电影评分的平均值。

年龄阶段划分

```
users = users.withColumn("ageGroup", F.when(F.col("age") < 18, 1)
    .when((F.col("age") >= 18) & (F.col("age") <= 24),
```

2)

```
.when((F.col("age") >= 25) & (F.col("age") <= 29),
```

3)

```

4) .when((F.col("age") >= 30) & (F.col("age") <= 34),
5) .when((F.col("age") >= 35) & (F.col("age") <= 39),
6) .when((F.col("age") >= 40) & (F.col("age") <= 49),
   .when(F.col("age") >= 50, 7).otherwise(0))

```

```
users.createOrReplaceTempView("user_with_age_group")
```

计算每个年龄阶段的平均评分

```

age_rating_avg = sqlContext.sql("""
    SELECT u.ageGroup, AVG(r.rating) AS avg_rating
    FROM user_with_age_group u
    JOIN rating r ON u.userId = r.userId
    GROUP BY u.ageGroup
    ORDER BY u.ageGroup
""")
age_rating_avg.show()

```

```

# 年龄阶段划分
users = users.withColumn("ageGroup", F.when(F.col("age") < 18, 1)
    .when((F.col("age") >= 18) & (F.col("age") <= 24), 2)
    .when((F.col("age") >= 25) & (F.col("age") <= 29), 3)
    .when((F.col("age") >= 30) & (F.col("age") <= 34), 4)
    .when((F.col("age") >= 35) & (F.col("age") <= 39), 5)
    .when((F.col("age") >= 40) & (F.col("age") <= 49), 6)
    .when(F.col("age") >= 50, 7).otherwise(0))

users.createOrReplaceTempView("user_with_age_group")

# 计算每个年龄阶段的平均评分
age_rating_avg = sqlContext.sql("""
    SELECT u.ageGroup, AVG(r.rating) AS avg_rating
    FROM user_with_age_group u
    JOIN rating r ON u.userId = r.userId
    GROUP BY u.ageGroup
    ORDER BY u.ageGroup
""")
age_rating_avg.show()

```



3) 接上题：统计每种年龄阶段对于每种体裁电影的偏好程度（每种体裁显示一个数量即可）。

```
age_genre_preference = sqlContext.sql("""
SELECT u.ageGroup, SUM(r.rating * f.action) AS action_score, SUM(r.rating * f.comedy) AS comedy_score
FROM user_with_age_group u
JOIN rating r ON u.userId = r.userId
JOIN film f ON r.movieId = f.movieId
GROUP BY u.ageGroup
ORDER BY u.ageGroup
""")
age_genre_preference.show()
```

| ageGroup | action_score | comedy_score |
|----------|--------------|--------------|
| 1 | 2615.0 | 2475.0 |
| 2 | 23613.0 | 24583.0 |
| 3 | 19617.0 | 22273.0 |
| 4 | 12771.0 | 14475.0 |
| 5 | 9627.0 | 11964.0 |
| 6 | 12545.0 | 14716.0 |
| 7 | 8268.0 | 10766.0 |

4) 统计不同**性别**对于所有电影评分的平均值。

```
gender_rating_avg = sqlContext.sql("""
SELECT u.gender, AVG(r.rating) AS avg_rating
FROM user u
JOIN rating r ON u.userId = r.userId
GROUP BY u.gender
""")
gender_rating_avg.show()
```

| gender | avg_rating |
|--------|--------------------|
| F | 3.5315073815073816 |
| M | 3.5292889846485322 |

5) 统计不同**性别**对于每种体裁电影的偏好程度（每种体裁显示一个数量即可）。

6) 统计不同**职业**对于所有电影评分的平均值。

7) 统计不同**职业**对于每种体裁电影的偏好程度（每种体裁显示一个数量即可）。

5 6 7 三道小题代码 和结果如下：

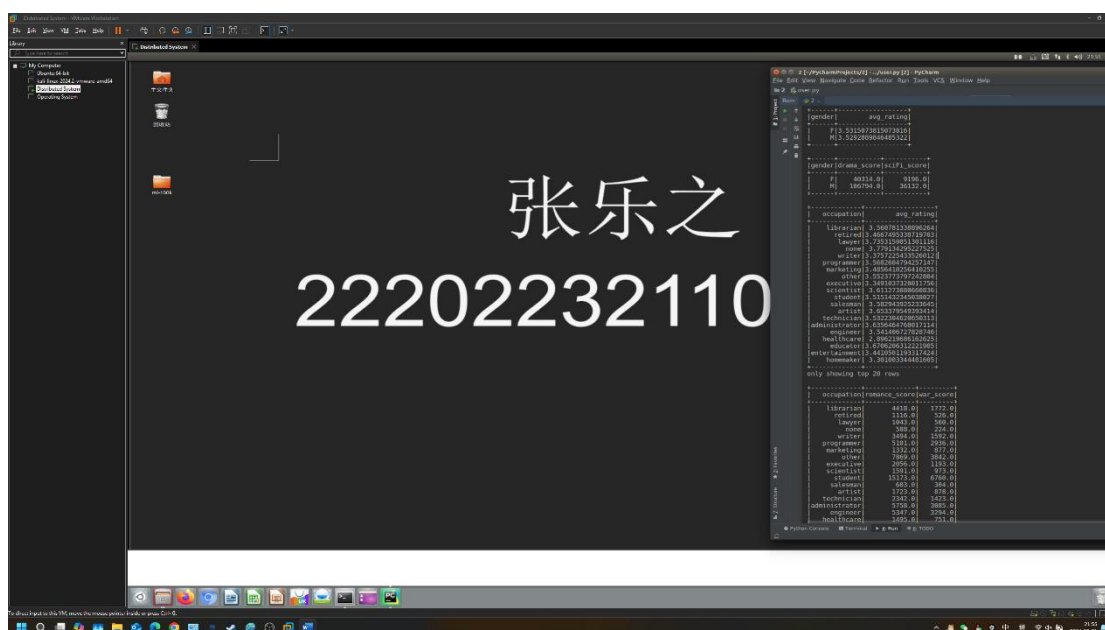
```

gender_genre_preference = sqlContext.sql("""
SELECT u.gender, SUM(r.rating * f.drama) AS drama_score, SUM(r.rating * f.sciFi) AS sciFi_score
FROM user u
JOIN rating r ON u.userId = r.userId
JOIN film f ON r.movieId = f.movieId
GROUP BY u.gender
""")
gender_genre_preference.show()

occupation_rating_avg = sqlContext.sql("""
SELECT u.occupation, AVG(r.rating) AS avg_rating
FROM user u
JOIN rating r ON u.userId = r.userId
GROUP BY u.occupation
""")
occupation_rating_avg.show()

occupation_genre_preference = sqlContext.sql("""
SELECT u.occupation, SUM(r.rating * f.romance) AS romance_score, SUM(r.rating * f.war) AS war_score
FROM user u
JOIN rating r ON u.userId = r.userId
JOIN film f ON r.movieId = f.movieId
GROUP BY u.occupation
""")
occupation_genre_preference.show()

```



说明：请务必将 1) -7) 综合分析以后，确定如何对数据进行预处理（类似系统设计），并设计相应代码或 SQL 语句，或综合应用各类方法实现统计分析任务。切忌不要看一题就马上做一题。

7、选择合适的算法实现推荐

1) 训练模型

```

als = ALS(maxIter=10, regParam=0.1, userCol="userId", itemCol="movieId", ratingCol="rating")
model = als.fit(ratings)

```

2) 评价模型

```

predictions = model.transform(ratings)
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating", predictionCol="prediction")
rmse = evaluator.evaluate(predictions)

```

Root Mean Square Error = 0.7724098834509139

Process finished with exit code 0

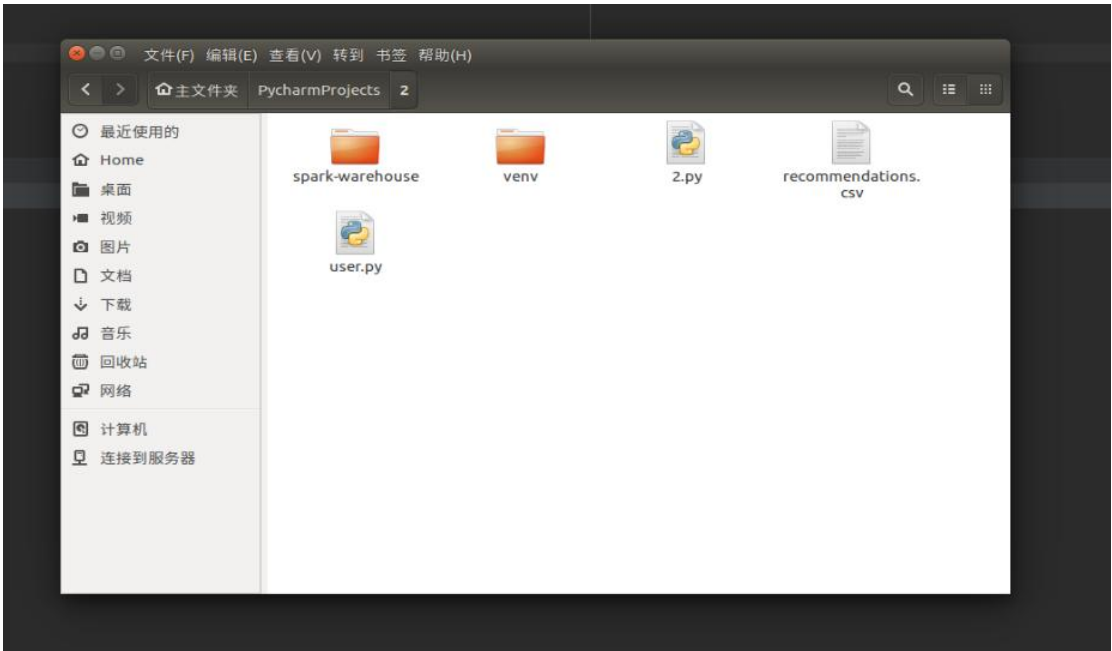
3) 推荐结果存储到文件


```

# 将推荐结果转换为 Pandas DataFrame 进行处理
user_recommendations_pd = user_recommendations.toPandas()

# 保存推荐结果为 CSV 文件
user_recommendations_pd.to_csv("./recommendations.csv", index=False)

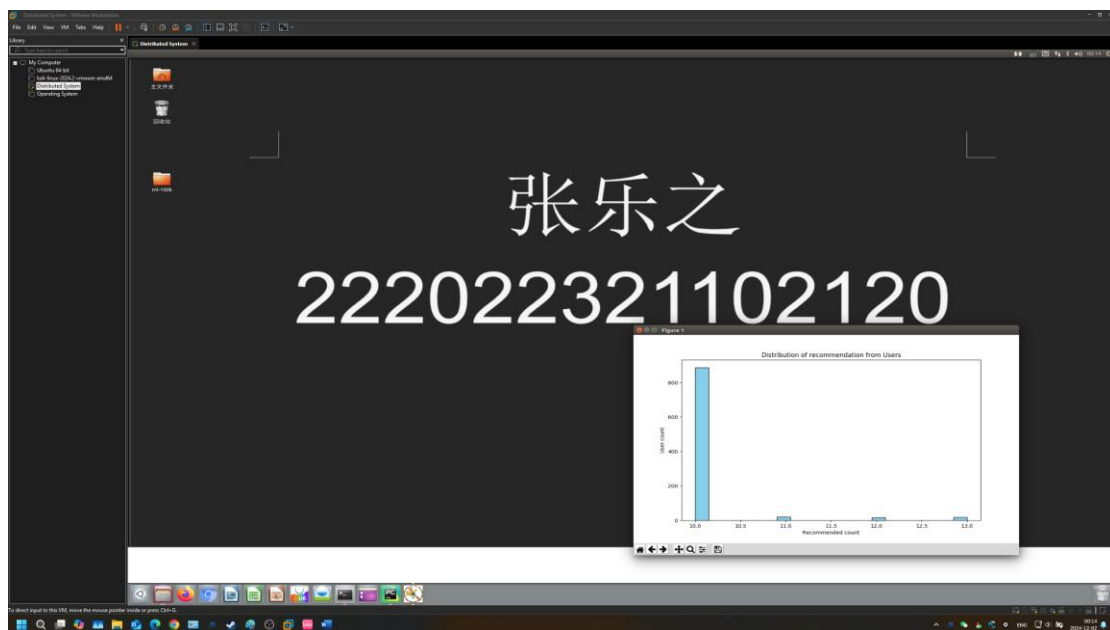
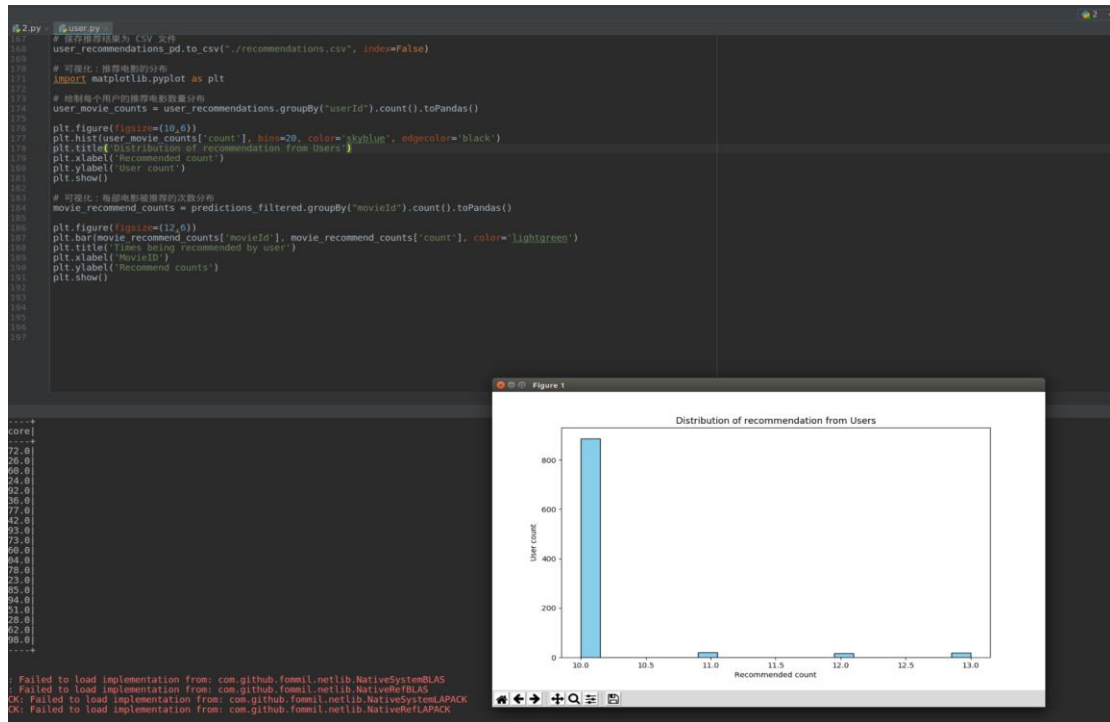
```

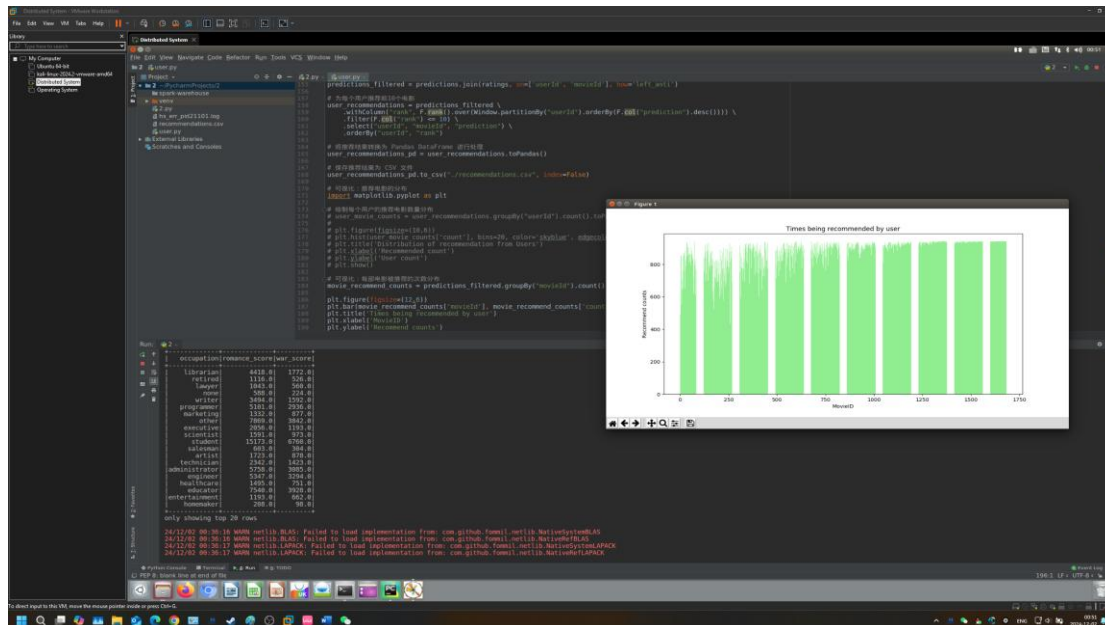


The screenshot shows an Excel spreadsheet titled 'recommendations.csv - Excel'. The spreadsheet has three columns: 'userid', 'movieid', and 'prediction'. The data is organized into rows, with the first row being the header and subsequent rows containing numerical values. The 'userid' column contains values ranging from 1 to 31. The 'movieid' column contains values ranging from 1449 to 1463. The 'prediction' column contains values ranging from 5.02043 to 4.027577.

| userid | movieid | prediction |
|--------|---------|------------|
| 1 | 1449 | 5.02043 |
| 1 | 408 | 5.000753 |
| 1 | 1367 | 4.96117 |
| 1 | 919 | 4.660414 |
| 1 | 512 | 4.61587 |
| 1 | 1524 | 4.604227 |
| 1 | 1467 | 4.603549 |
| 1 | 1122 | 4.583725 |
| 1 | 511 | 4.581613 |
| 1 | 474 | 4.534952 |
| 2 | 1449 | 4.970369 |
| 2 | 113 | 4.711959 |
| 2 | 408 | 4.702795 |
| 2 | 1398 | 4.632488 |
| 2 | 64 | 4.616422 |
| 2 | 318 | 4.613372 |
| 2 | 1643 | 4.612265 |
| 2 | 119 | 4.611915 |
| 2 | 515 | 4.596463 |
| 2 | 190 | 4.58699 |
| 3 | 1643 | 4.895717 |
| 3 | 1268 | 4.764369 |
| 3 | 1607 | 4.392355 |
| 3 | 1143 | 4.339308 |
| 3 | 42 | 4.31364 |
| 3 | 838 | 4.179735 |
| 3 | 1467 | 4.174956 |
| 3 | 1664 | 4.114118 |
| 3 | 902 | 4.109481 |
| 3 | 1463 | 4.027577 |

4) 推荐结果可视化





总结：

在本次实验中，我完成了基于 **Spark** 的电影推荐系统构建任务，利用 **ALS**（交替最小二乘法）算法进行电影推荐，并通过可视化和性能评估对模型效果进行了全面分析。整个实验涉及了数据加载、预处理、模型训练与优化以及结果评估与展示多个环节，全面提升了我对大数据处理和机器学习模型应用的理解。

在数据预处理阶段，我加载并清洗了来自 **MovieLens** 数据集的用户、电影以及评分数据，处理了数据中的缺失值和格式问题，并将其转化为 **Spark DataFrame** 进行后续分析。通过对用户评分数据的分析，提取了用户偏好的信息，构建了训练数据集，为后续的推荐模型提供了可靠的输入。

接着，我利用 **Spark MLlib** 中的 **ALS** 算法构建了电影推荐模型，训练了包含用户、电影和评分的矩阵因式分解模型。通过调参优化了模型的性能，利用交叉验证和 **RMSE**（均方根误差）评估了模型的准确性，并进行了模型的评估与优化。在训练过程中，我进一步理解了推荐系统如何通过矩阵分解技术捕捉用户和物品之间的潜在关系，从而为用户推荐个性化内容。

为了实现个性化推荐，我手动实现了用户电影推荐列表的生成，通过对预测评分的排序为每个用户推荐前 **10** 部电影。这一过程加深了我对推荐算法的理解，并能够根据每个用户的历史评分数据，提供精准的电影推荐。

最后，我使用 **Python** 的 **Pandas** 和 **Matplotlib** 对推荐结果进行了可视化展示。通过绘制用户评分分布、电影类别分布以及推荐结果的可视化图表，我将推荐系统的效果直观地呈现给用户，提升了分析的可解释性和用户体验。

此次实验让我全面掌握了基于 **Spark** 的推荐系统实现方法，深入理解了推荐算法的核心思想与实际应用。通过本次实践，我不仅加深了对大数据处理、分布式计算和机器学习模型的理解，还提升了数据预处理、模型评估与优化、可视化展示等方面的能力。这个实验为我今后在推荐系统领域的学习与应用提供了宝贵的经验。

参考资料及提示：

- 1) <https://dmlab.xmu.edu.cn/blog/1781/#more-1781>

- 2) https://blog.csdn.net/2401_84052244/article/details/140913550
- 3) https://blog.51cto.com/u_14457/11936443