Research of Anime Recommendation 2020

Group 48

Jiayi Yang 512470
Name sid
Tong Wang 515400
Yen-Cheng Chen 511210

Table of Content

EXECUTIVE SUMMARY

This report studies the factors that affect anime scores, and builds a prediction model for anime scores based on the dataset "Anime Recommendation 2020" using linear regression. The big data tools used include Pyspark, Mapreduce, Hive and Impala. All the code used and the final model can be found in the appendix. This report first briefly introduces the background and overall content of the study. Next, the problem statement introduces the process of problem identification and proposes the main research questions of this paper. Chapter 3 introduces the information of the dataset selected for this report, and explains why Big Data and this dataset are used for research. The next chapter introduces the methodology of this report, which mainly includes the following four parts: 1. Preliminary screening of factors by screening the correlation between columns and scores of different datasets; 2. Processing datasets, removing NA values, and converting data 3. Build the model; 4. Display the results and model accuracy. Finally, in Chapter 5, we summarize the obtained models and factors. The limitations of the research project are critically analyzed to give practical recommendations.

CHAPTER 1 INTRODUCTION

After entering the information age, film and television works have received dividends from the Internet, and using the Internet as a platform has greatly increased their scope of communication. At the same time, this also means that film and television works in the new era will be subject to more rigorous scrutiny. Especially when the scoring website appeared, the audience had a way to express their subjective opinions on the works. However, this isn't necessarily good news for these ratings for producers. Low ratings are a significant blow to film and television works, especially anime, because its audience is more active on the Internet and is easily affected by low ratings on the Internet. Therefore, creating a high-rated anime has become a matter of great concern to anime producers.

Based on the database "Anime recommendation 2020", this report is dedicated to studying factors that affect anime ratings. By using big data tools such as Mapreduce, Hive, pyspark, and Impala, build a model to predict animation scores based on known data. The reminding parts of the report will begin with a problem statement, providing a well description of the main question searched. Next, chapter 3 introduces the details of the selected dataset and the codes used. Methodology of the research is discussed in Chapter 4 and the final results and discussion. Finally, in Chapter 5, the conclusion, suggestions and limitations of the research are given.

CHAPTER 2 PROBLEM STATEMENT

2.1 Problem Description

Anime producers are worried about the rating risk of their own products because ratings will not only affect the playback volume of work, causing profit fluctuations, but also have a potential impact on the company's image. Based on this concern, our team builds an anime rating prediction model with reference to the animation recommendation data in 2020. The aim of this report is to provide anime producers with a reliable animation rating prediction tool using the factors we have, to help producers predict the possible ratings of the works before the anime is released, so as to avoid the potential risks faced by anime producers when they release their works.

In addition, we also want to conduct research on the current public's preference for anime. Animation is a rapidly growing branch of the film and television industry, with a very mature market in countries such as the United States and Japan. With the development of technology, the emergence of online platforms has further promoted the development of the animation industry, along with many potential business opportunities. This group is very interested in exploring these opportunities by mining data.

2.2 Research Question

The main research question for this report is to build up an anime score prediction model. To provide reference for anime producers, identifying which factors and how they affect the score of a specific anime.

CHAPTER 3 CODE AND DATASET

3.1 Data Description

This database is set for anime recommendations that are shown in year 2020. It is collected in year 2021 and contains two parts data, the anime data and user preference data. The anime data part is provided by MyAnimeList, and the user preference part is produced by Jikan API. We found this dataset on Kaggle website. The whole dataset is about 693MB as a zip file and 2.87 GB in total. The link for this dataset is https://www.kaggle.com/datasets/hernan4444/anime-recommendation-database-2020. We primarily use the anime.csv file in the zip file downloaded which contains multiple related csv files. It has 17562 anime records, 35 feature columns (such as score, genres studios, popularity, members, favorites and plan to watch) and the preference from 325772 different users in total. It is a semi-structured dataset in CSV format so we need to make some necessary adjustments to the database to make it well-structured for data analysis.

3.2 Why Big Data

3.2.1 Dataset

There are two main reasons for choosing "Anime Recommendation 2020" as the research dataset.

Factor Diversity

There are many columns in this data set, covering all aspects of an anime, which is convenient for us to include more factors when building a model.

Timeliness
This dataset is the data of 2020, and the release time is very close to now. Therefore, the audience's preferences and anime trends are not much different from the current ones. Data is time-sensitive.

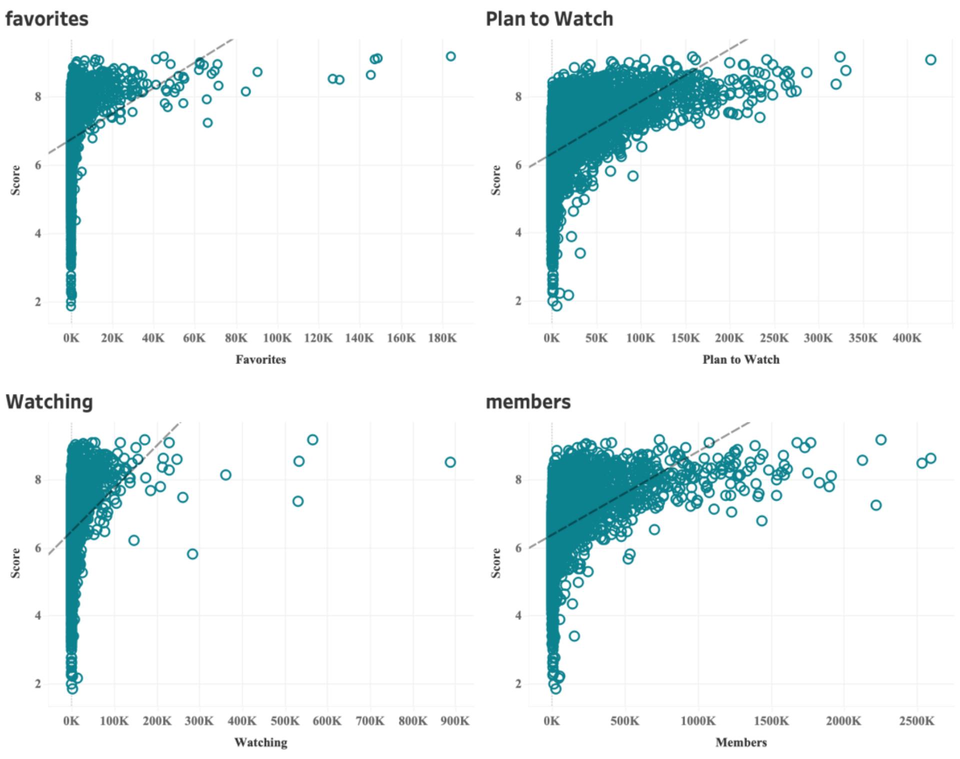
3.2.2 Big Data
3V Principle
The Anime Recommendation 2020 database meets all the characteristics of the 3V principle that big data has.

- Volume: The entire database contains five csv files that record information related to the audience ratings the animation received from different aspects, as well as an html folder that stores the raw data of the comments. The total amount of data is 2.87GB, which can be called a big data in terms of volume.
- Velocity: Big data is often accompanied by fleeting business opportunities. It is important for companies to get insight through data in a timely manner and to seize the opportunity to apply it. For the database selected for this report, it records data and information of animation reviews between February 26th and March 20th in 2020. Broadcast platforms and animation producers can only achieve their business goals of increasing traffic and profits if they seize the right moment to recommend great works to their audience.
- Variety: The database contains many different types of data, including strings, numbers, time, etc. Although it has been classified by column based on data types, there are cases where the data in the same column has inconsistent units, which in turn increases the data diversity.

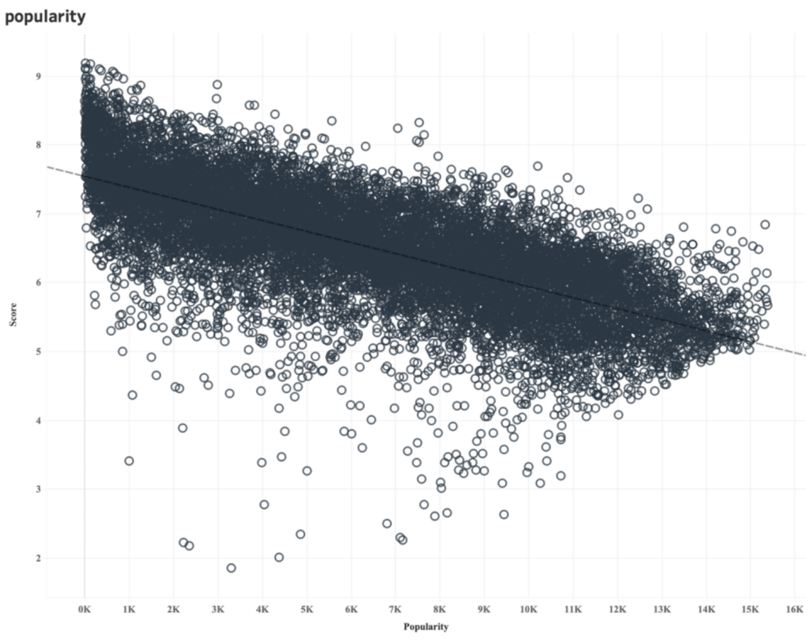
Processing Difficulty
Besides, inability to use traditional processes and tools to quickly analyze data in databases to aid in business decisions. For example, the anmielist.csv file contains more data than will fit on a single worksheet and can’t be opened in on the personal laptop. Hence there’s need for big data and cloud computing tools.

CHAPTER 4 METHODOLOGY

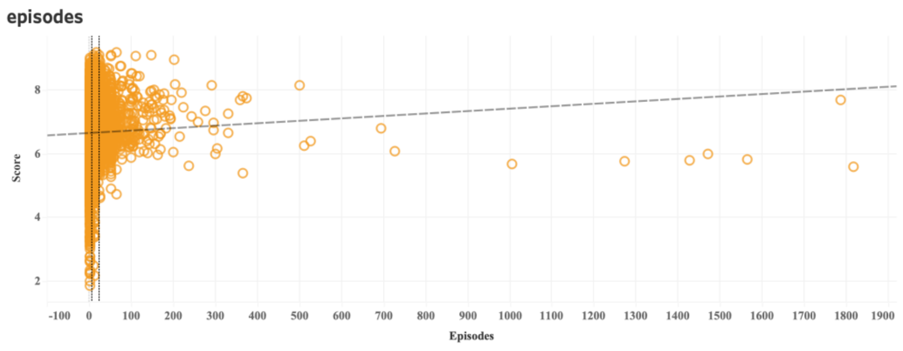
4.1 Variable Screening
First, we compare “Favorites”, “Plan to Watch”, “Watching”, and “Members” with “Score” and make scatter plot for each using tableau. As we expect, these attributes all have positive correlation with “Score”, which means when the amount for these attributes increase, the more likely they give higher scores to the anime.

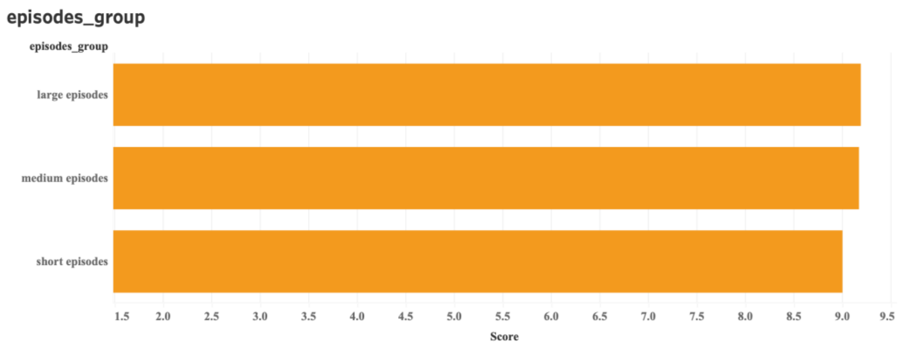


Next, we do the same thing on “popularity” to compare with “Score”. According to the plot, it shows that there is obvious negative correlation between these two, which is a surprising result. That means when there is more popularity with the anime, the score could decrease.

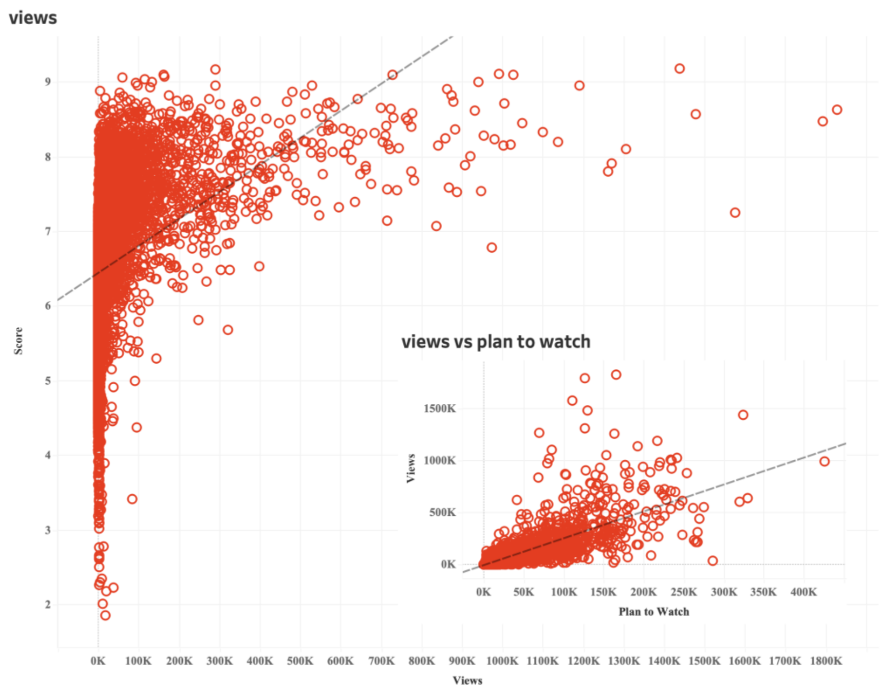


And then we go on to look at the attribute “Episodes”. We do the scatter plot as before, but the results below for the upper plot does not look obvious, which R-square value is 0.003, so we decide to divide to three groups: “short episodes”, “medium episodes”, and “large episodes”, and compared those with average scores to see if there is any pattern. The lower bar plot using impala and tableau shows the results. For the results, it also shows no obvious pattern that the averages for each group are:9.19,9.17, and 9.00. As a result, we decide to remove this attribute in the following model.



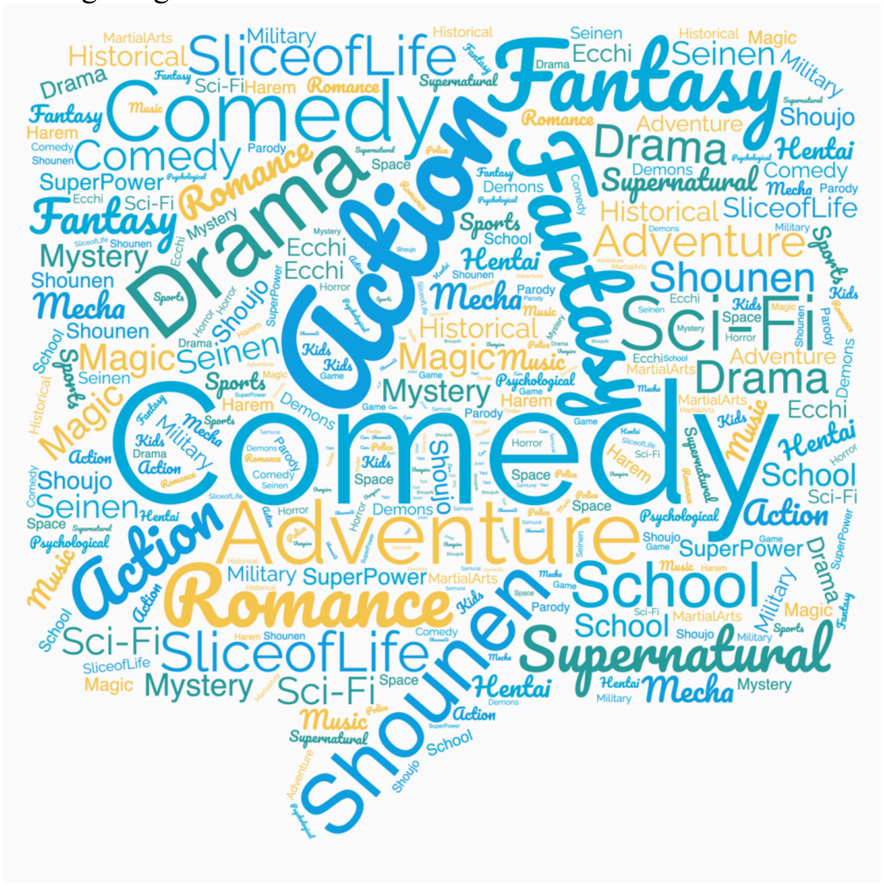


Last, we take a look at the new column “Views” we created and compared it to “Scores”. Before this, we compared “Plan to watch” to “Views” to see if there is any correlation between these two. We expect there will be high positive correlation between these two, but if the correlation is too high, we should not add “Views” into the following model. Fortunately, the R-spared value is 0.64 which is not over 0.7, although it somehow affects a little on results, it could still be an important attribute, so we will keep this attribute. For the bigger plots for the results, it shows positive correlation between “Views” and “Scores”.



We used word count including MapReduce technique to describe “genre” attribute because each anime has multiple types of genres. We filtered out the genres above the total average score which is 6.51, and then split the data, count each type of genre to get the word cloud.

According to the results, comedy and action are the most common types of genre among the anime which get higher scores.



4.2 Data sourcing & Data processing

Data Source:
myanimelist.net. A collected database with information about 17.562 anime and the preference from 325.772 different users between February 26th and March 20th in 2020.

Data Processing
The pre-processing of the raw data aims at structuring the data and resolving missing values and outliers. Clean and tidy structured data is obtained before building the model.

- Step 1: Dealing with NA value
  It is necessary to have complete data information when building the model. The missing data are concentrated in “Score”, “Source”, “Rating” these three columns and they are marked as “unknown”. We cannot simply replace the missing data with mean values, zeros or the most common category, because each animation work is unique and it is not logical in reality to simply make assumptions. Therefore, for the data in the database, we choose to remove the rows with missing values and do not consider those in further into the study.
- Step 2: Transform Data Type
  Inspection of the data reveals that some variables that should be numbers, such as “Scores”, “Popularity” appear as strings in the original data and need to be converted to the desired numeric type first.
- Step 3: Create A New Column
  We want to know if the actual total number of people scoring will have an impact on the Scores, but the relevant data is not available from the raw data and needs to be processed by a simple calculation. Create a new column, add up the number of

people who scored 1-10 respectively from the raw data, and define it as “Views”, representing the total number of comments a particular animation received.

- Step 4: Choose the Column
  The choice of too many variables in a linear regression model may affect the results of the model and overfitting problems may occur. On the basis of ensuring the predictive accuracy of the model, we selected the variables considered to have the most significant impact on the scores to build the model. A total of 11 variables were selected, which contained three categorical variables and eight numerical variables.
- Step 5: Convert Categorical Data
  Some categorical data that are useful to study the Scoring model of an animation are Type, Source and Rating. These are important characteristics to describe an animation and appear as strings in the raw data. When measuring these variables in the model, they need to be first converted into numbers, and each type will be randomly defined as a fixed number to facilitate the calculation in the model.

4.3 Model Construction & Description

Variables Selection

- Categorical Variables (X): Type, Source, Rating
- Numerical Variables (X): Popularity, Members, Favorites, Watching, On-Hold, Dropped, Plan to Watch, Views
- Dependent Variable (Y): Score

Subset Selection

By setting a seed, 70% of the data are randomly selected from the database as training data to build the model, and the remaining 30% of the data are used as test data for testing and prediction.

Model Selection

Linear Regression: The score is the dependent variable Y that we need to predict, which is a continuous variable, i.e., what we are looking at is a combination of mutually independent factors that affect the scoring situation. The model built out is a primary functional relationship between the variables.

$$Y = \alpha + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_3 X_3 + \varepsilon$$

4.4 Results

Error Rate

Mean Squared Error	R Squared Error
0.3685942532977168	0.5320844620570304

Prediction Example

Features	Prediction
[9448.0,1577.0,1....	5.898949599246521
[9409.0,1597.0,1....	5.905856513538575
[9967.0,1249.0,4....	5.845432882437087
[8817.0,2013.0,1...	6.208256058983739
[997.0,149948.0,2…	6.422304324758633
…	…

CHAPTER 5 CONCLUSION

5.1 Conclusion

After the above investigation, we successfully obtained a predictive model code about anime scores. You only need to output the corresponding x variable to get the predicted score. (Appendix 1) We finally concluded that the factors that affect anime scores are: Type, Source, Rating, Popularity, Members, Favorites, Watching, On-Hold, Dropped, Plan to Watch, and Views. Favorites, Plan to Watch, Watching, Members, and Views are positively correlated with the score; Popularity is negatively correlated with the score.

基于这个结果，我们建议(基于全部的 factor 趋势对 producer 提出建议)

5.2 Critical Analysis, Limitations and Future Research

Overall, the modeling was successful. We have a relatively small error rate, and the overall prediction accuracy is in line with what would be expected for a linear regression model. More importantly, the linear regression model has the advantage of being highly interpretable. Through programming, we can look directly at the coefficients in front of the independent variables to determine in what way each variable affects the dependent variable.

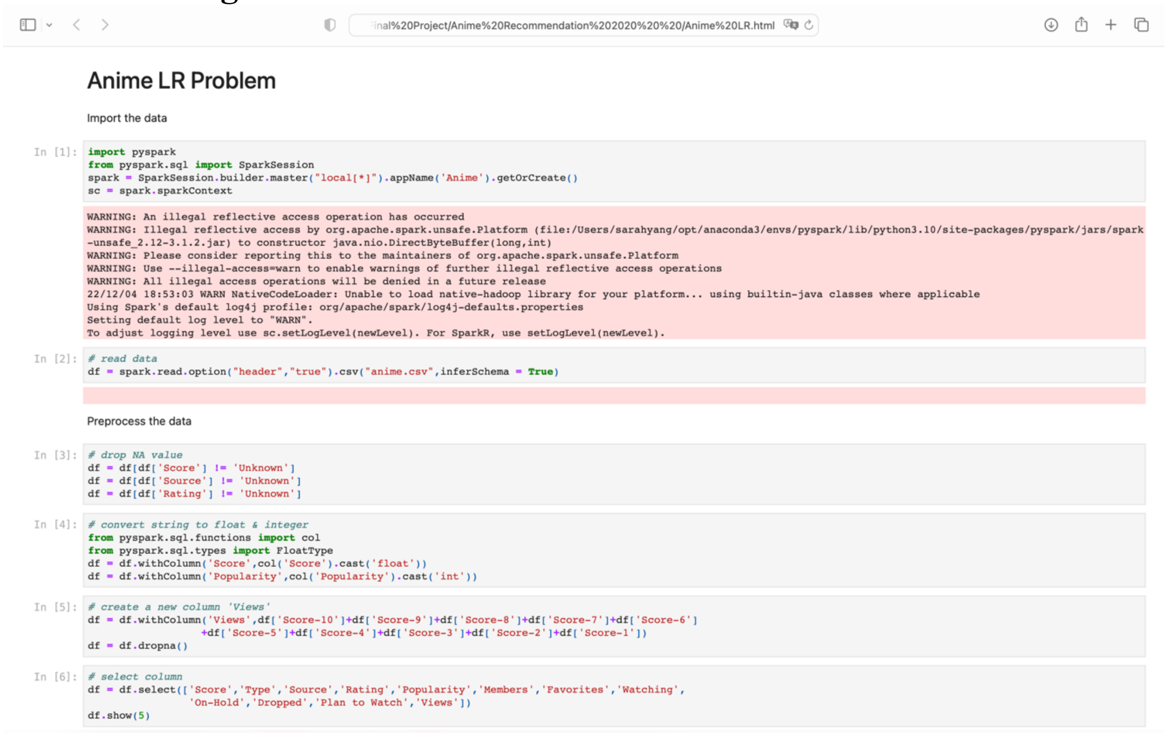
However, there are still many limitations in this research. First, Each animation viewer has their own unique preferences, and it is debatable whether to put the data of all animation works into the same model when building the model. Further experimentation is needed to see if the prediction results are more accurate when classifying animation by different criteria, such as by type or by studio. Since special cases need to be taken into account, such as restricted animation with a limited audience, further filtering and adjustment is needed when making recommendations for a specific audience.

In addition, the classical linear regression model has relatively strict assumptions. However, the current model does not fully meet these requirements. For example, whether the X variables are completely independent of each other. Is there a potential linear relationship between the number of people who are watching the animation and the number of people who have dropped? We need to consider problems like multicollinearity. Whether the errors of the data in the database conform to a normal distribution? In the future research, we need to investigate whether the data meet these assumptions and make further adjustments further.

APPENDIX

Appendix 1: Codes

● Linear Regression



inal%20Project/Anime%20Recommendation%202020%20%20/Anime%20LR.html

Anime LR Problem

Import the data

```
In [1]: import pyspark
        from pyspark.sql import SparkSession
        spark = SparkSession.builder.master("local[*]").appName('Anime').getOrCreate()
        sc = spark.sparkContext
```

```
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.spark.unsafe.Platform (file:/Users/sarahyang/opt/anaconda3/envs/pyspark/lib/python3.10/site-packages/pyspark/jars/spark
-unsafe_2.12-3.1.2.jar) to constructor java.nio.DirectByteBuffer(long,int)
WARNING: Please consider reporting this to the maintainers of org.apache.spark.unsafe.Platform
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
22/12/04 18:53:03 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
```

```
In [2]: # read data
        df = spark.read.option("header","true").csv("anime.csv",inferSchema = True)
```

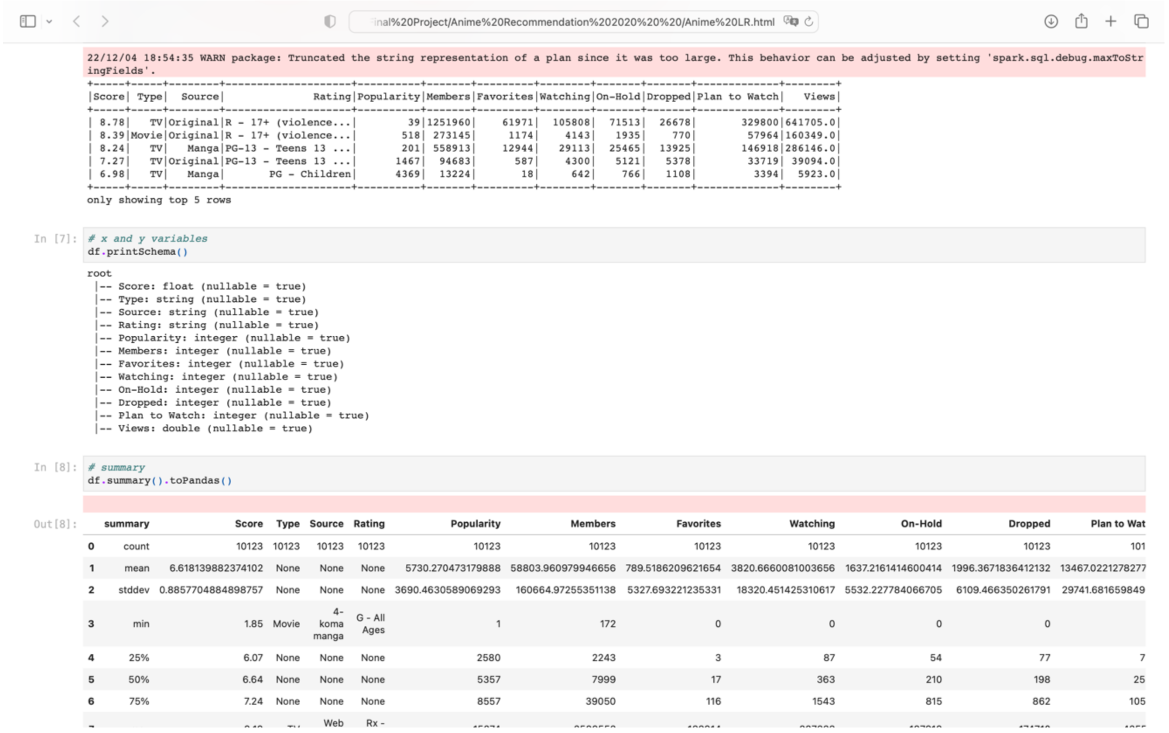
Preprocess the data

```
In [3]: # drop NA value
        df = df[df['Score'] != 'Unknown']
        df = df[df['Source'] != 'Unknown']
        df = df[df['Rating'] != 'Unknown']
```

```
In [4]: # convert string to float & integer
        from pyspark.sql.functions import col
        from pyspark.sql.types import FloatType
        df = df.withColumn('Score',col('Score').cast('float'))
        df = df.withColumn('Popularity',col('Popularity').cast('int'))
```

```
In [5]: # create a new column 'Views'
        df = df.withColumn('Views',df['Score-10']+df['Score-9']+df['Score-8']+df['Score-7']+df['Score-6']
                           +df['Score-5']+df['Score-4']+df['Score-3']+df['Score-2']+df['Score-1'])
        df = df.dropna()
```

```
In [6]: # select column
        df = df.select(['Score','Type','Source','Rating','Popularity','Members','Favorites','Watching',
                        'On-Hold','Dropped','Plan to Watch','Views'])
        df.show(5)
```



inal%20Project/Anime%20Recommendation%202020%20%20/Anime%20LR.html

```
22/12/04 18:54:35 WARN package: Truncated the string representation of a plan since it was too large. This behavior can be adjusted by setting 'spark.sql.debug.maxToStr
ingFields'.
+-----+-----+--------+--------------------+----------+-------+---------+--------+-------+-------+-------------+--------+
|Score| Type|  Source|              Rating|Popularity|Members|Favorites|Watching|On-Hold|Dropped|Plan to Watch|   Views|
+-----+-----+--------+--------------------+----------+-------+---------+--------+-------+-------+-------------+--------+
| 8.78|   TV|Original|R - 17+ (violence...|        39|1251960|    61971|  105808|  71513|  26678|       329800|641705.0|
| 8.39|Movie|Original|R - 17+ (violence...|       518| 273145|     1174|    4143|   1935|    770|        57964|160349.0|
| 8.24|   TV|   Manga|PG-13 - Teens 13 ...|       201| 558913|    12944|   29113|  25465|  13925|       146918|286146.0|
| 7.27|   TV|Original|PG-13 - Teens 13 ...|      1467|  94683|      587|    4300|   5121|   5378|        33719| 39094.0|
| 6.98|   TV|   Manga|       PG - Children|      4369|  13224|       18|     642|    766|   1108|         3394|  5923.0|
+-----+-----+--------+--------------------+----------+-------+---------+--------+-------+-------+-------------+--------+
only showing top 5 rows
```

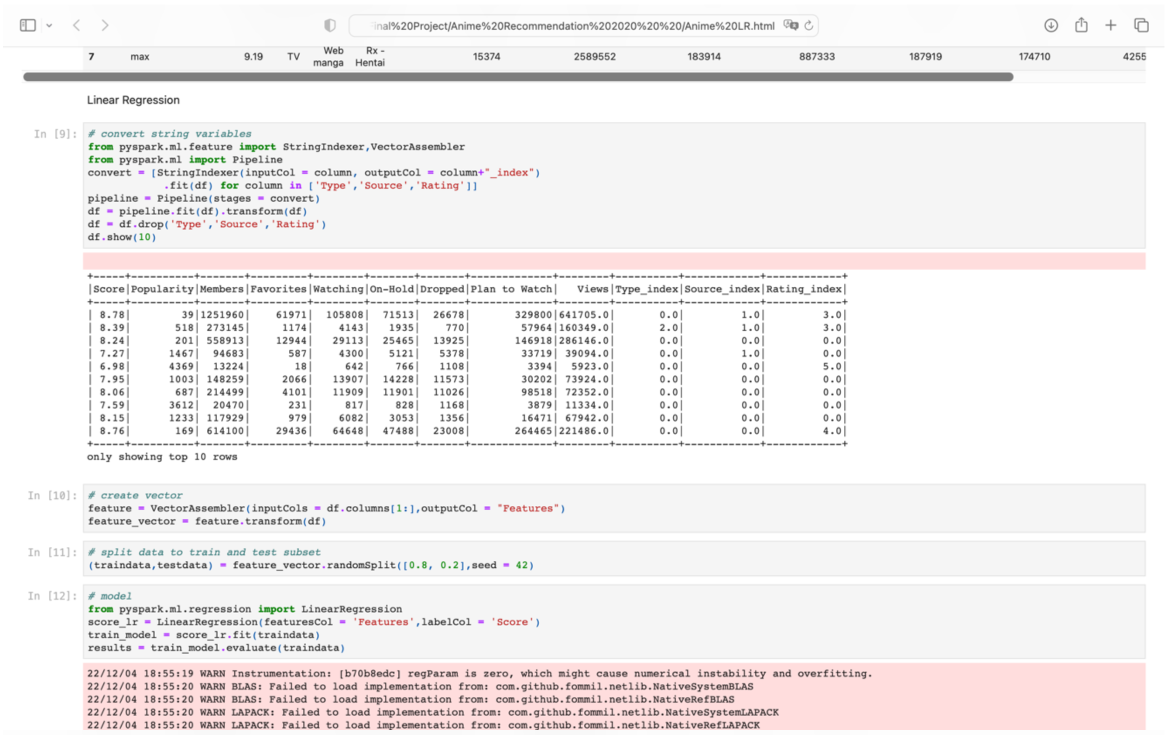
```
In [7]: # x and y variables
        df.printSchema()
```

```
root
 |-- Score: float (nullable = true)
 |-- Type: string (nullable = true)
 |-- Source: string (nullable = true)
 |-- Rating: string (nullable = true)
 |-- Popularity: integer (nullable = true)
 |-- Members: integer (nullable = true)
 |-- Favorites: integer (nullable = true)
 |-- Watching: integer (nullable = true)
 |-- On-Hold: integer (nullable = true)
 |-- Dropped: integer (nullable = true)
 |-- Plan to Watch: integer (nullable = true)
 |-- Views: double (nullable = true)
```

```
In [8]: # summary
        df.summary().toPandas()
```

Out[8]:

	summary	Score	Type	Source	Rating	Popularity	Members	Favorites	Watching	On-Hold	Dropped	Plan to Wat
0	count	10123	10123	10123	10123	10123	10123	10123	10123	10123	10123	101
1	mean	6.618139882374102	None	None	None	5730.270473179888	58803.960979946656	789.5186209621654	3820.6660081003656	1637.2161414600414	1996.3671836412132	13467.0221278277
2	stddev	0.8857704884898757	None	None	None	3690.4630589069293	160664.97255351138	5327.693221235331	18320.451425310617	5532.227784066705	6109.466350261791	29741.681659849
3	min	1.85	Movie	4-koma manga	G - All Ages	1	172	0	0	0	0	
4	25%	6.07	None	None	None	2580	2243	3	87	54	77	7
5	50%	6.64	None	None	None	5357	7999	17	363	210	198	25
6	75%	7.24	None	None	None	8557	39050	116	1543	815	862	105
[illegible]	[illegible]	[illegible]	[illegible]	Web	Rx -	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]



```
7    max    9.19    TV    Web manga    Rx - Hentai    15374    2589552    183914    887333    187919    174710    4255
```

Linear Regression

```
In [9]:
# convert string variables
from pyspark.ml.feature import StringIndexer,VectorAssembler
from pyspark.ml import Pipeline
convert = [StringIndexer(inputCol = column, outputCol = column+"_index")
            .fit(df) for column in ['Type','Source','Rating']]
pipeline = Pipeline(stages = convert)
df = pipeline.fit(df).transform(df)
df = df.drop('Type','Source','Rating')
df.show(10)
```

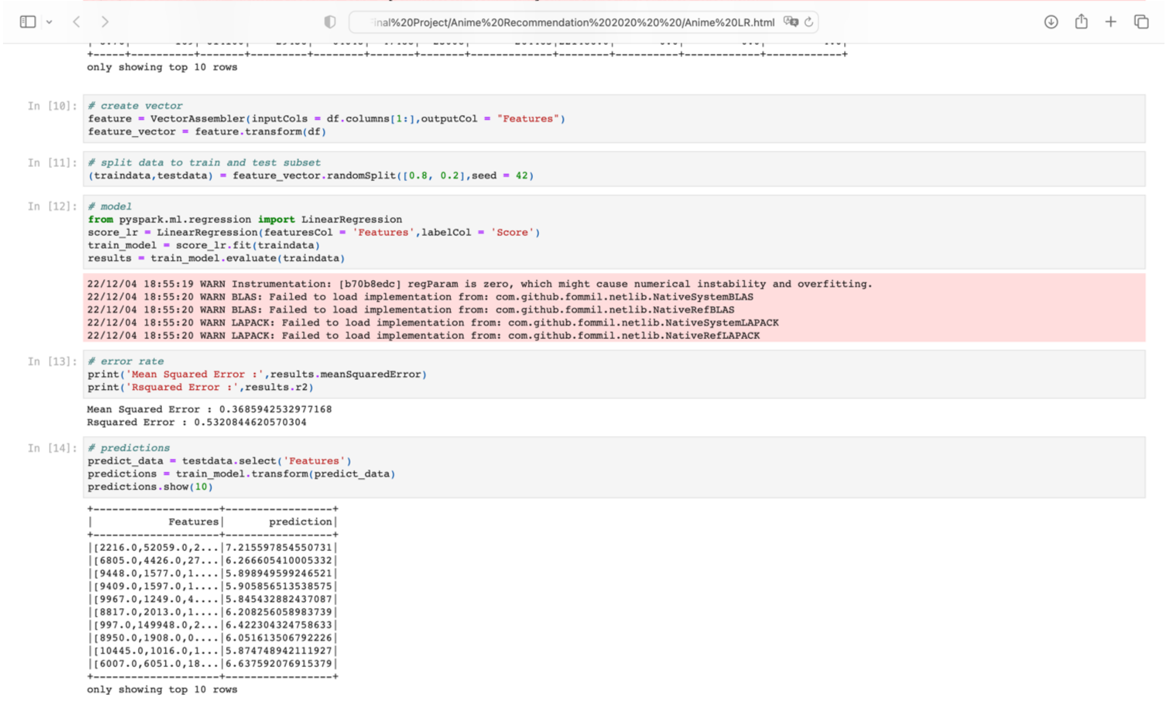
```
+-----+----------+-------+---------+--------+-------+-------+-------------+--------+----------+------------+------------+
|Score|Popularity|Members|Favorites|Watching|On-Hold|Dropped|Plan to Watch|   Views|Type_index|Source_index|Rating_index|
+-----+----------+-------+---------+--------+-------+-------+-------------+--------+----------+------------+------------+
| 8.78|        39|1251960|    61971|  105808|  71513|  26678|       329800|641705.0|       0.0|         1.0|         3.0|
| 8.39|       518| 273145|     1174|    4143|   1935|    770|        57964|160349.0|       2.0|         1.0|         3.0|
| 8.24|       201| 558913|    12944|   29113|  25465|  13925|       146918|286146.0|       0.0|         0.0|         0.0|
| 7.27|      1467|  94683|      587|    4300|   5121|   5378|        33719| 39094.0|       0.0|         1.0|         0.0|
| 6.98|      4369|  13224|       18|     642|    766|   1108|         3394|  5923.0|       0.0|         0.0|         5.0|
| 7.95|      1003| 148259|     2066|   13907|  14228|  11573|        30202| 73924.0|       0.0|         0.0|         0.0|
| 8.06|       687| 214499|     4101|   11909|  11901|  11026|        98518| 72352.0|       0.0|         0.0|         0.0|
| 7.59|      3612|  20470|      231|     817|    828|   1168|         3879| 11334.0|       0.0|         0.0|         0.0|
| 8.15|      1233| 117929|      979|    6082|   3053|   1356|        16471| 67942.0|       0.0|         0.0|         0.0|
| 8.76|       169| 614100|    29436|   64648|  47488|  23008|       264465|221486.0|       0.0|         0.0|         4.0|
+-----+----------+-------+---------+--------+-------+-------+-------------+--------+----------+------------+------------+
only showing top 10 rows
```

```
In [10]:
# create vector
feature = VectorAssembler(inputCols = df.columns[1:],outputCol = "Features")
feature_vector = feature.transform(df)
```

```
In [11]:
# split data to train and test subset
(traindata,testdata) = feature_vector.randomSplit([0.8, 0.2],seed = 42)
```

```
In [12]:
# model
from pyspark.ml.regression import LinearRegression
score_lr = LinearRegression(featuresCol = 'Features',labelCol = 'Score')
train_model = score_lr.fit(traindata)
results = train_model.evaluate(traindata)
```

```
22/12/04 18:55:19 WARN Instrumentation: [b70b8edc] regParam is zero, which might cause numerical instability and overfitting.
22/12/04 18:55:20 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeSystemBLAS
22/12/04 18:55:20 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeRefBLAS
22/12/04 18:55:20 WARN LAPACK: Failed to load implementation from: com.github.fommil.netlib.NativeSystemLAPACK
22/12/04 18:55:20 WARN LAPACK: Failed to load implementation from: com.github.fommil.netlib.NativeRefLAPACK
```



```
only showing top 10 rows
```

```
In [10]:
# create vector
feature = VectorAssembler(inputCols = df.columns[1:],outputCol = "Features")
feature_vector = feature.transform(df)
```

```
In [11]:
# split data to train and test subset
(traindata,testdata) = feature_vector.randomSplit([0.8, 0.2],seed = 42)
```

```
In [12]:
# model
from pyspark.ml.regression import LinearRegression
score_lr = LinearRegression(featuresCol = 'Features',labelCol = 'Score')
train_model = score_lr.fit(traindata)
results = train_model.evaluate(traindata)
```

```
22/12/04 18:55:19 WARN Instrumentation: [b70b8edc] regParam is zero, which might cause numerical instability and overfitting.
22/12/04 18:55:20 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeSystemBLAS
22/12/04 18:55:20 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeRefBLAS
22/12/04 18:55:20 WARN LAPACK: Failed to load implementation from: com.github.fommil.netlib.NativeSystemLAPACK
22/12/04 18:55:20 WARN LAPACK: Failed to load implementation from: com.github.fommil.netlib.NativeRefLAPACK
```

```
In [13]:
# error rate
print('Mean Squared Error :',results.meanSquaredError)
print('Rsquared Error :',results.r2)
```

```
Mean Squared Error : 0.3685942532977168
Rsquared Error : 0.5320844620570304
```

```
In [14]:
# predictions
predict_data = testdata.select('Features')
predictions = train_model.transform(predict_data)
predictions.show(10)
```

```
+--------------------+-----------------+
|            Features|       prediction|
+--------------------+-----------------+
|[2216.0,52059.0,2...|7.215597854550731|
|[6805.0,4426.0,27...|6.266605410005332|
|[9448.0,1577.0,1....|5.898949599246521|
|[9409.0,1597.0,1....|5.905856513538575|
|[9967.0,1249.0,4....|5.845432882437087|
|[8817.0,2013.0,1....|6.208256058983739|
|[997.0,149948.0,2...|6.422304324758633|
|[8950.0,1908.0,0....|6.051613506792226|
|[10445.0,1016.0,1...|5.874748942111927|
|[6007.0,6051.0,18...|6.637592076915379|
+--------------------+-----------------+
only showing top 10 rows
```

```
Anime LR Problem
Import the data

import pyspark
from pyspark.sql import SparkSession
spark =
SparkSession.builder.master("local[*]").appName('Anime').getOrCreate()
sc = spark.sparkContext
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.spark.unsafe.Platform
(file:/Users/sarahyang/opt/anaconda3/envs/pyspark/lib/python3.10/site-
packages/pyspark/jars/spark-unsafe_2.12-3.1.2.jar) to constructor
java.nio.DirectByteBuffer(long,int)
```

```
WARNING: Please consider reporting this to the maintainers of
org.apache.spark.unsafe.Platform
WARNING: Use --illegal-access=warn to enable warnings of further illegal
reflective access operations
WARNING: All illegal access operations will be denied in a future release
22/12/04 18:53:03 WARN NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-
defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).
# read data
df = spark.read.option("header","true").csv("anime.csv",inferSchema =
True)
```

Preprocess the data

```
# drop NA value
df = df[df['Score'] != 'Unknown']
df = df[df['Source'] != 'Unknown']
df = df[df['Rating'] != 'Unknown']
# convert string to float & integer
from pyspark.sql.functions import col
from pyspark.sql.types import FloatType
df = df.withColumn('Score',col('Score').cast('float'))
df = df.withColumn('Popularity',col('Popularity').cast('int'))
# create a new column 'Views'
df = df.withColumn('Views',df['Score-10']+df['Score-9']+df['Score-
8']+df['Score-7']+df['Score-6']
                  +df['Score-5']+df['Score-4']+df['Score-3']+df['Score-
2']+df['Score-1'])
df = df.dropna()
# select column
df =
df.select(['Score','Type','Source','Rating','Popularity','Members','Favorit
es','Watching',
                'On-Hold','Dropped','Plan to Watch','Views'])
df.show(5)
22/12/04 18:54:35 WARN package: Truncated the string representation of a
plan since it was too large. This behavior can be adjusted by setting
'spark.sql.debug.maxToStringFields'.
+-----+-----+--------+--------------------+----------+-------+---------+---
-----+-------+-------+-------------+--------+
|Score|
Type|   Source|               Rating|Popularity|Members|Favorites|Watching|On
-Hold|Dropped|Plan to Watch|   Views|
+-----+-----+--------+--------------------+----------+-------+---------+---
-----+-------+-------+-------------+--------+
| 8.78|   TV|Original|R - 17+
(violence...|         39|1251960|    61971|  105808|  71513|  26678|       3
29800|641705.0|
| 8.39|Movie|Original|R - 17+ (violence...|       518|
273145|      1174|     4143|    1935|     770|         57964|160349.0|
| 8.24|    TV|    Manga|PG-13 - Teens 13 ...|       201|
558913|     12944|    29113|   25465|   13925|        146918|286146.0|
| 7.27|    TV|Original|PG-13 - Teens 13
...|        1467|  94683|       587|     4300|    5121|    5378|         33719|
39094.0|
| 6.98|    TV|    Manga|        PG -
Children|      4369|  13224|        18|      642|     766|    1108|          339
4|  5923.0|
+-----+-----+--------+------------------+----------+-------+---------+---
-----+-------+-------+-------------+--------+
only showing top 5 rows

# x and y variables
df.printSchema()
root
 |-- Score: float (nullable = true)
 |-- Type: string (nullable = true)
 |-- Source: string (nullable = true)
 |-- Rating: string (nullable = true)
 |-- Popularity: integer (nullable = true)
 |-- Members: integer (nullable = true)
 |-- Favorites: integer (nullable = true)
 |-- Watching: integer (nullable = true)
 |-- On-Hold: integer (nullable = true)
 |-- Dropped: integer (nullable = true)
 |-- Plan to Watch: integer (nullable = true)
 |-- Views: double (nullable = true)

# summary
df.summary().toPandas()


summary Score    Type    Source  Rating  Popularity  Members
Favorites    Watching     On-Hold Dropped Plan to Watch    Views
0    count    10123    10123    10123    10123    10123    10123    10123    101
23    10123    10123    10123    10123
1    mean    6.6181398823741 02    None    None    None    5730.27047317988
8    58803.960979946656    789.5186209621654    3820.6660081003656    1637.2161
414600414    1996.3671836412132    13467.022127827719    30493.593796305442
2    stddev   0.8857704884898757    None    None    None    3690.46305890692
93    160664.97255351138    5327.693221235331    18320.451425310617    5532.2277
84066705    6109.466350261791    29741.68165984968    96417.33167399192
3    min    1.85    Movie    4-koma manga    G - All Ages    1    172
0    0    0    0    13    101.0
4    25%
6.07    None    None    None    2580    2243    3    87    54    77    789
745.0
5    50%    6.64    None    None    None    5357    7999    17    363    210
198    2509    3110.0
6    75%    7.24    None    None    None    8557    39050    116
1543    815    862    10587    17351.0
```

```
7   max 9.19    TV  Web manga   Rx - Hentai 15374   2589552
183914  887333  187919  174710  425531  1826691.0
Linear Regression

# convert string variables
from pyspark.ml.feature import StringIndexer,VectorAssembler
from pyspark.ml import Pipeline
convert = [StringIndexer(inputCol = column, outputCol = column+"_index")
            .fit(df) for column in ['Type','Source','Rating']]
pipeline = Pipeline(stages = convert)
df = pipeline.fit(df).transform(df)
df = df.drop('Type','Source','Rating')
df.show(10)


+-----+----------+-------+---------+--------+-------+-------+------------
+--------+----------+------------+------------+
|Score|Popularity|Members|Favorites|Watching|On-Hold|Dropped|Plan to
Watch|   Views|Type_index|Source_index|Rating_index|
+-----+----------+-------+---------+--------+-------+-------+------------
+--------+----------+------------+------------+
|
8.78|        39|1251960|    61971|  105808|  71513|  26678|       329800|64
1705.0|        0.0|           1.0|           3.0|
| 8.39|       518|
273145|      1174|     4143|    1935|     770|        57964|160349.0|       2.0
|         1.0|           3.0|
| 8.24|       201|
558913|     12944|    29113|   25465|   13925|       146918|286146.0|       0.0
|         0.0|           0.0|
| 7.27|      1467|  94683|      587|    4300|   5121|   5378|        33719|
39094.0|        0.0|           1.0|           0.0|
|
6.98|      4369|  13224|       18|     642|    766|   1108|         3394|
5923.0|        0.0|           0.0|           5.0|
| 7.95|      1003| 148259|     2066|   13907|  14228|  11573|        30202|
73924.0|        0.0|           0.0|           0.0|
.     | 8.06|       687|
214499|      4101|    11909|   11901|   11026|        98518|
72352.0|        0.0|           0.0|           0.0|
.     |
7.59|      3612|  20470|      231|     817|    828|   1168|         3879|
11334.0|        0.0|           0.0|           0.0|
.     | 8.15|      1233|
117929|       979|     6082|    3053|    1356|        16471|
67942.0|        0.0|           0.0|           0.0|
.     | 8.76|       169|
614100|     29436|    64648|   47488|   23008|       264465|221486.0|       0.0
|         0.0|           4.0|
.     +-----+----------+-------+---------+--------+-------+-------+--------
-----+--------+----------+------------+------------+
.     only showing top 10 rows
.      
.     # create vector
.      feature = VectorAssembler(inputCols = df.columns[1:],outputCol = 
"Features")
.      feature_vector = feature.transform(df)
.      # split data to train and test subset
.      (traindata,testdata) = feature_vector.randomSplit([0.8, 0.2],seed = 
42)
.      # model
.      from pyspark.ml.regression import LinearRegression
.      score_lr = LinearRegression(featuresCol = 'Features',labelCol = 
'Score')
.      train_model = score_lr.fit(traindata)
.      results = train_model.evaluate(traindata)
.      22/12/04 18:55:19 WARN Instrumentation: [b70b8edc] regParam is 
zero, which might cause numerical instability and overfitting.
.      22/12/04 18:55:20 WARN BLAS: Failed to load implementation from: 
com.github.fommil.netlib.NativeSystemBLAS
.      22/12/04 18:55:20 WARN BLAS: Failed to load implementation from: 
com.github.fommil.netlib.NativeRefBLAS
.      22/12/04 18:55:20 WARN LAPACK: Failed to load implementation from: 
com.github.fommil.netlib.NativeSystemLAPACK
.      22/12/04 18:55:20 WARN LAPACK: Failed to load implementation from: 
com.github.fommil.netlib.NativeRefLAPACK
.      # error rate
.      print('Mean Squared Error :',results.meanSquaredError)
.      print('Rsquared Error :',results.r2)
.      Mean Squared Error : 0.3685942532977168
.      Rsquared Error : 0.5320844620570304
.      # predictions
.      predict_data = testdata.select('Features')
.      predictions = train_model.transform(predict_data)
.      predictions.show(10)
.      +--------------------+------------------+
.      |            Features|        prediction|
.      +--------------------+------------------+
.      |[2216.0,52059.0,2...|7.215597854550731|
.      |[6805.0,4426.0,27...|6.266605410005332|
.      |[9448.0,1577.0,1....|5.898949599246521|
.      |[9409.0,1597.0,1....|5.905856513538575|
.      |[9967.0,1249.0,4....|5.845432882437087|
.      |[8817.0,2013.0,1....|6.208256058983739|
.      |[997.0,149948.0,2...|6.422304324758633|
.      |[8950.0,1908.0,0....|6.051613506792226|
.      |[10445.0,1016.0,1...|5.874748942111927|
.      |[6007.0,6051.0,18...|6.637592076915379|
.      +--------------------+------------------+
.      only showing top 10 rows
```

Appendix 2: Table

Appendix 3: ~~Marking criteria~~

Appendix 4 - ~~Peer assessment~~