Research of Anime Recommendations 2020

Group 48

Jiayi Yang 512470
Tong Wang 515400
Yen-Cheng Chen 511210
Zihao Zhao 515512

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 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 problem identification process and proposes this paper's main research questions. 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 dramatically 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 could express their subjective opinions on the works. However, this is not 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. 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 studies factors that affect anime ratings. 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 sound description of the main question searched. Next, chapter 3 introduces the details of the selected dataset and the codes used. The research methodology is discussed in Chapter 4, and the final results and discussion. Finally, in Chapter 5, the research's conclusion, suggestions, and limitations are given.

CHAPTER 2 PROBLEM STATEMENT

2.1 Problem Description

Anime producers are worried about the rating risk of their products because ratings will not only affect the playback volume of work, causing profit fluctuations but also potentially impact the company's image. Based on this concern, our team built an anime rating prediction model about the animation recommendation data in 2020. This report aims 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, avoiding the potential risks they face when they release their works.

In addition, we also want to research the current public's preference for anime. Animation is a rapidly growing 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 a solid tool 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 was collected in 2021 and contained two parts: the anime and user preference data. MyAnimeList provides the anime data part, and Jikan API produces the user preference part. We found this dataset on the Kaggle website. The whole dataset is about 693MB as a zip file, 2.87 GB as raw data in total with all information included, and 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 17,562 anime records, 35 feature columns (such as score, genre studios, popularity, members, favorites, and plan to watch), and the preference from 325,772 different users. 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 similar to 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, called big data in terms of volume.
- **Velocity**: Fleeting business opportunities often accompany big data. It is essential for companies to get insight through data promptly and to seize the opportunity to apply it. The database selected for this report records data and information on 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 data types, 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 have inconsistent units, increasing the data diversity.

Processing Difficulty
Besides, needing to use traditional processes and tools to analyze data in databases to aid in business decisions quickly. For example, the anmielist.csv file contains more data than will fit on a single worksheet and cannot be opened on a personal laptop. Hence there is a 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 a scatter plot for each using tableau. As we expect, these attributes all have a positive correlation with "Score", which means when the amount for these attributes increases, the more likely they give higher scores to the anime.

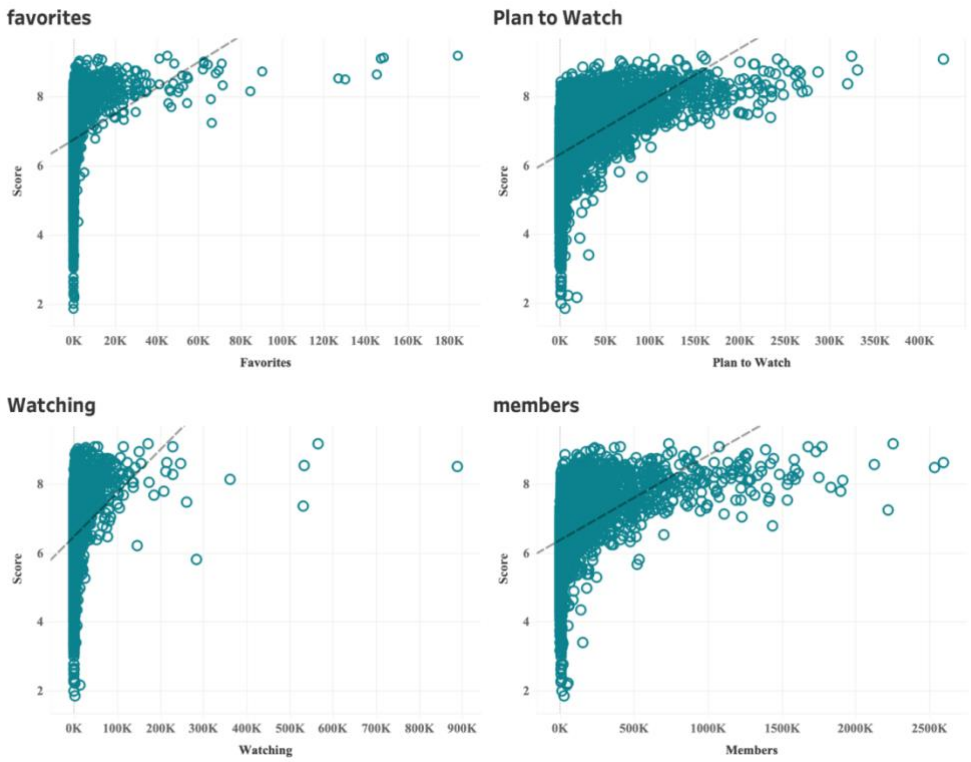


Figure 1 Scatter Plot for "Favorites", "Plan to Watch", "Watching" and "Members"

Next, we do the same thing on "Popularity" to compare with "Score". According to the plot, there is an apparent negative correlation between these two, which is a surprising result. That means the score could decrease when there is more popularity with the anime.

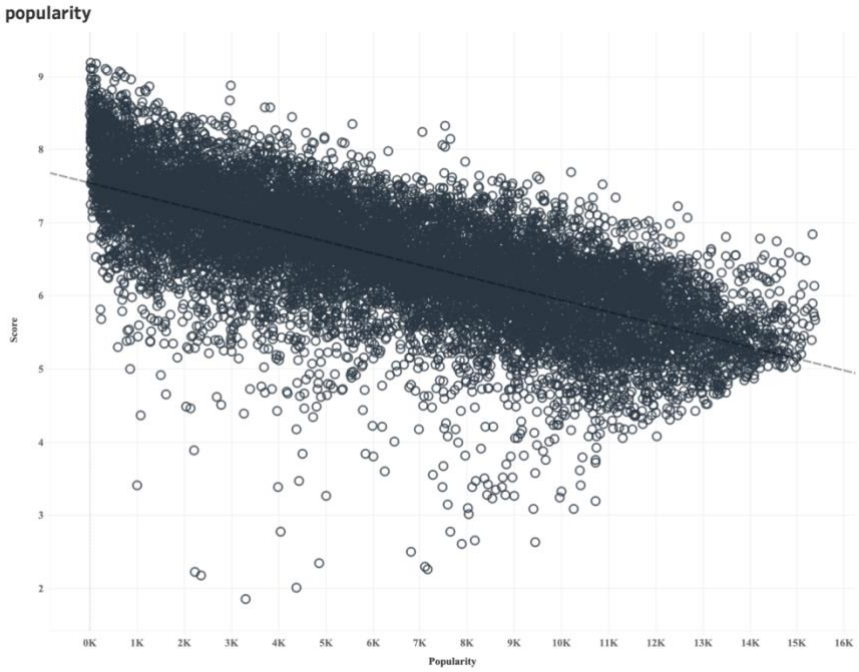


Figure 2 Scatter Plot for "Popularity"

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 do not look obvious, which R-square value is 0.003, so we decide to divide into 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. The results also show 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 and Episodes will not be considered in the model.

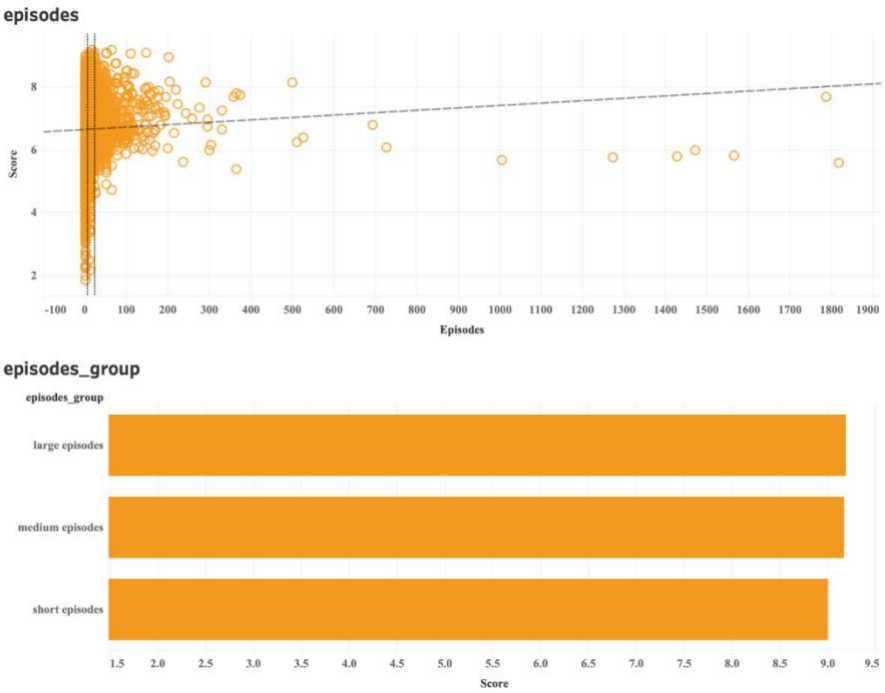


Figure 3 Scatter Plot for "Episodes" & Bar Plot for "short episodes", "medium episodes" and "large episodes"

Next, we looked 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 a 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 them a little on results, it could still be an essential attribute so that we will keep it. The bigger plots for the results show a positive correlation between "Views" and "Scores".

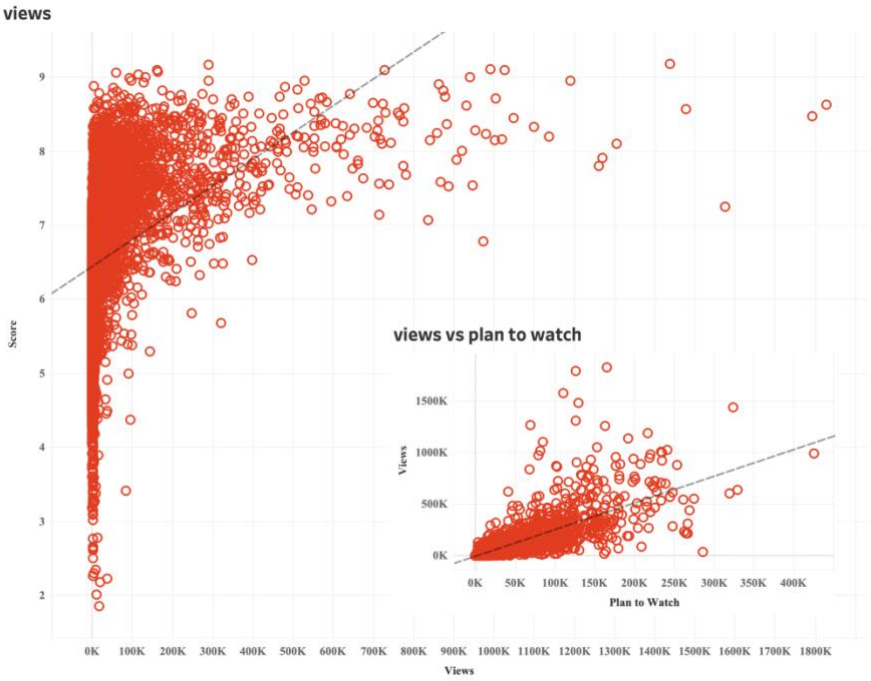
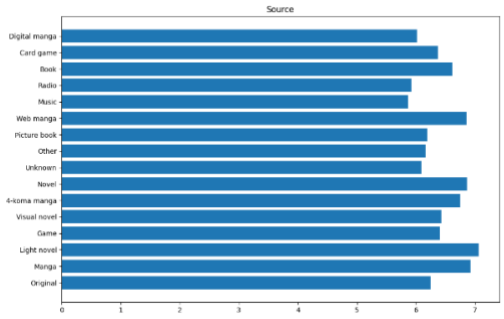
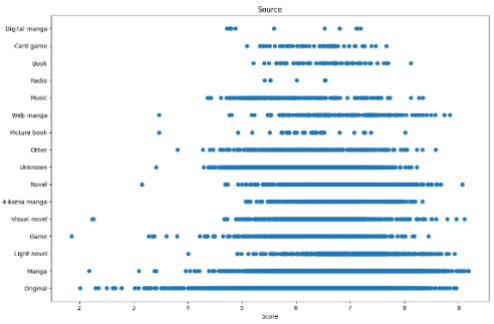
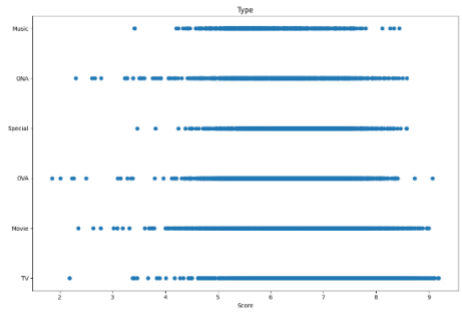


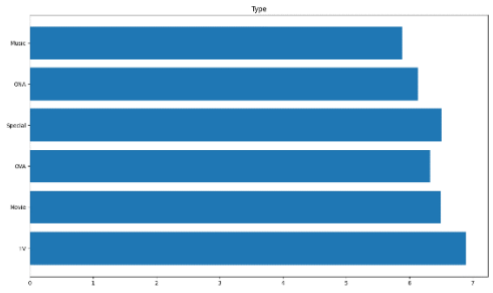
Figure 4 Correlation between "Views" and "Scores"

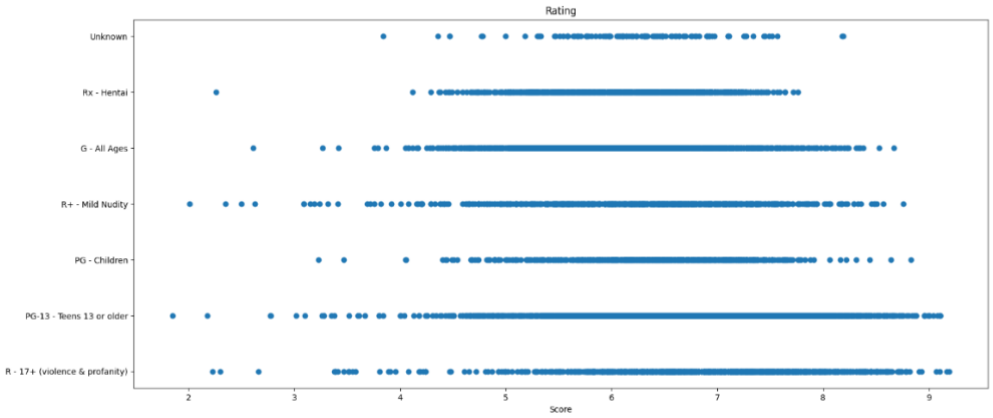
Source, Type, Rating & Premiered

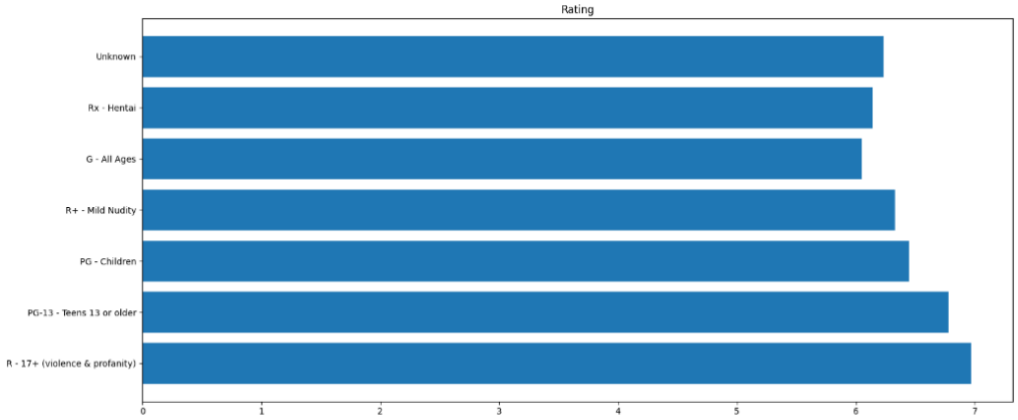


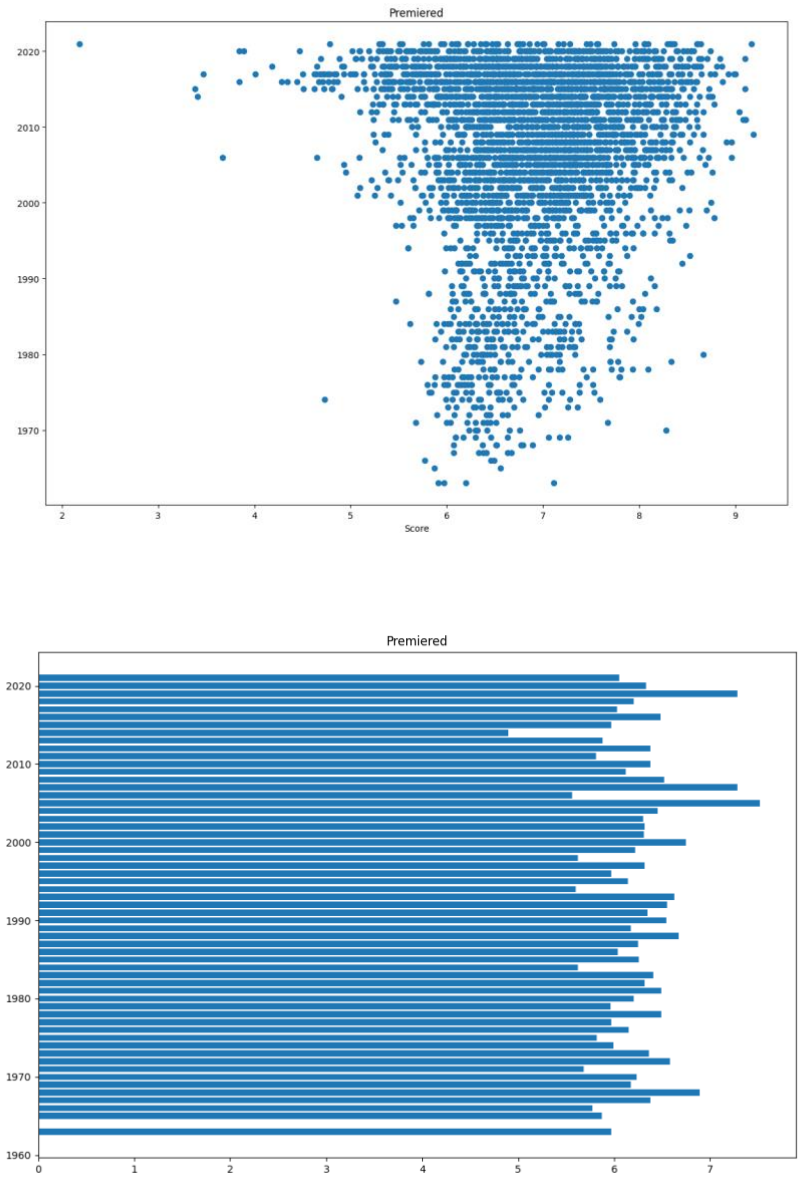












Figures 5 Categorical Variables

For the categorical variables, it is apparent that specific categories such as the light novel, R 17+ (violence & profanity), TV and premiered in the 21st century give the anime an advantage in getting a higher score. Thus, it is worth including them in our linear regression model as predictors.

We used word count, including the MapReduce technique, to describe the "genre" attribute because each anime has multiple types of genres. We filtered out the genres above the average score 6.51, then split the data and counted each type of genre to get the word cloud. According to the results, comedy and action are the most common genres in anime, which get higher scores.

Figure 6 "Genre" Word Cloud

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 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" and "Rating" these three columns 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 to make assumptions. Therefore, for the data in the database, we choose to remove the rows with missing values and do not consider those further into the study.
- Step 2: Transform Data Type
  Inspection of the data reveals that some variables that should be numbered, 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 total number of people scoring will impact 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 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 model's results, and overfitting problems may occur. To ensure 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 are valuable to study the Scoring model of animation are "Type", "Source", and "Rating". These are essential 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. 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.

Based on the results, we recommend the producer to pay attention to the high popularity anime, since the score shows a surprising negative correlation with the popularity which contradicts with our common sense. High popularity cannot always lead to a good score. Besides, anime that has source from light novel and rating of violence and profanity usually has higher score.

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 aligns with what would be expected for a linear regression model. More importantly, the linear regression model has the advantage of being highly interpretable. We can look directly at the coefficients in front of the independent variables through programming to determine how each variable affects the dependent variable.

However, there are still many limitations in this research. First, Each animation viewer has 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 type or studio. Since exceptional cases need to be considered, such as restricted animation with a limited audience, further filtering and adjustment are needed when making recommendations for a specific audience.

In addition, the classical linear regression model has relatively strict assumptions. However, the current model only partially meets these requirements. For example, whether the X variables are entirely independent of each other. Is there a potential linear relationship between the number of people watching the animation and the number of people who have dropped? Whether the errors of the data in the database conform to a normal distribution? In future research, we need to investigate whether the data meet these assumptions and make further adjustments.

APPENDIX

Appendix : Codes

- ## 1. Linear Regression

```
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
   10123   10123   10123   10123   10123   10123   10123
      1   mean    6.618139882374102   None    None    None    5730.27047
3179888   58803.960979946656   789.5186209621654   3820.6660081003656   163
7.2161414600414   1996.3671836412132   13467.022127827719   30493.5937963054
42
      2   stddev  0.8857704884898757  None    None    None    3690.46305
89069293  160664.97255351138  5327.693221235331   18320.451425310617  553
2.227784066705   6109.466350261791   29741.68165984968   96417.3316739919
2
      3   min 1.85    Movie   4-koma manga    G - All Ages    1   172
0   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
```

- **2. Word Cloud**

```
import pandas as pd
df=pd.read_csv('anime.csv')
df= df[df['Score'] != 'Unknown']
df['Score']=pd.to_numeric(df['Score'])
df= df[df['Score'] >6.51]
distinc=[]
for i in list(df['Genres']):
    lis=i.split(",")
    for j in lis:
        j=j.strip(" ")
        if j not in distinc:
            distinc.append(j)
dicgen={}
for i in distinc:
    dicgen[i]=0
for i in list(df['Genres']):
    lis=i.split(",")
    for j in lis:
        j=j.strip(" ")
        if j in distinc:
            dicgen[j]+=1
dicgen
Out[6]:
{'Action': 2048,
 'Adventure': 1416,
 'Comedy': 2761,
 'Drama': 1451,
 'Sci-Fi': 1235,
 'Space': 232,
 'Mystery': 490,
 'Shounen': 1347,
 'Police': 152,
 'Supernatural': 884,
 'Magic': 589,
 'Fantasy': 1407,
 'Sports': 344,
 'Josei': 65,
 'Romance': 1218,
 'Slice of Life': 885,
 'Cars': 29,
 'Seinen': 521,
 'Horror': 171,
 'Psychological': 217,
 'Thriller': 83,
 'Super Power': 354,
 'Martial Arts': 221,
 'School': 1006,
 'Ecchi': 394,
 'Vampire': 93,
 'Military': 345,
```

```
        'Historical': 445,
        'Dementia': 33,
        'Mecha': 509,
        'Demons': 252,
        'Samurai': 97,
        'Game': 151,
        'Harem': 283,
        'Music': 388,
        'Shoujo': 402,
        'Shoujo Ai': 55,
        'Shounen Ai': 57,
        'Kids': 235,
        'Hentai': 409,
        'Parody': 202,
        'Yaoi': 18,
        'Yuri': 8}
```

- **3. Figure 5's code**

```
import csv
import matplotlib.pyplot as plt

# read data
fp = open('anime.csv', 'rt', encoding='utf-8')
reader = csv.reader(fp)
#for i in reader:
#    if i[6] != 'Unknown':
#        data.append(i)
data = list(reader)
fp.close()

# create a directory
cols = {data[0][i]:i for i in range(len(data[0])) }
data.pop(0)

# scatter
def scatter(pdata, title):
    plt.scatter(pdata['x0'], pdata['y0'], marker='o')
    plt.title(title)
    plt.xlabel('Score')
    plt.show()

# barh
def barh(pdata, title):
    plt.barh(pdata['x1'], pdata['y1'])
    plt.title(title)
    plt.show()

# caculate x,y
def calc(col):
    global data
    global cols

    pdata = {'x0':[], 'y0':[], 'x1':[], 'y1':[]}
    av = {}

    for i in data:
        # pass the 'Unknown'
        if i[2] == 'Unknown':
            continue
        pdata['x0'].append(float(i[2]))
        pdata['y0'].append(i[cols[col]])

        if i[cols[col]] not in av:
            av.update({i[cols[col]]:[]})

        av[i[cols[col]]].append(float(i[2]))
```

```
            for i in av:
                pdata['x1'].append(i)
                pdata['y1'].append(sum(av[i]) / len(av[i]))

            return pdata

        # caculate x,y
        def calcP(col='Premiered'):
            global data
            global cols

            pdata = {'x0':[], 'y0':[], 'x1':[], 'y1':[]}
            av = {}

            for i in data:
                # pass the 'Unknown'
                if i[2] == 'Unknown':
                    continue

                if i[cols[col]] == 'Unknown':
                    continue

                pdata['x0'].append(float(i[2]))
                pdata['y0'].append(int(i[cols[col]][-4:]))

                if i[cols[col]][-4:] not in av:
                    av.update({int(i[cols[col]][-4:]):[]})

                av[int(i[cols[col]][-4:])].append(float(i[2]))

            tmp = av
            av = dict(sorted(av.items(), key=lambda x:x[0]))



            for i in av:
                pdata['x1'].append(i)
                pdata['y1'].append(sum(av[i]) / len(av[i]))

            return pdata

        # plot
        for i in ['Source', 'Rating', 'Type']:
            pd = calc(i)
            scatter(pd, i)
            barh(pd, i)

        pd = calcP()
        scatter(pd, 'Premiered')
```

```
.      barh(pd, 'Premiered')
```