Amy Ning

amy.ning1@baruchmail.cuny.edu

CIS 4130 CMWA

Prof. Holowczak

Proposal

The data set contains over 100 million user reviews from Steam. The data set can be found and downloaded on Kaggle: https://www.kaggle.com/datasets/kieranpoc/steam-reviews. There are a total of 24 columns, each row contains information on a review such as the text, the game title, and the rating.

List of data set attributes

- author
  - steamid
  - number of games owned
  - number of reviews
  - playtime all time
  - playtime over the last 2 weeks
  - playtime at the time of the review
  - when they last played the game
- language
- time created
- time updated
- if the review was positive or negative
- number of people who voted the review up
- number of people who voted the review funny
- a helpfulness score (steam generated)
- number of comments
- if the user purchased the game on Steam
- if the user checked a box saying they got the app for free
- if the user posted this review while the game was in Early Access

The project aims to predict the helpfulness score of Steam reviews. The helpfulness score indicates how useful the review is that can be determined by factors such as other users votes and number of comments. Linear regression will be used to predict the helpfulness scores of Steam reviews.

Data Acquisition (see Appendix B)

To collect the data for my project, I first downloaded the API token file(kaggle.json) from Kaggle.

Create a VM instance on GCP with 100 GB for the boot disk.

Launch the shell and create a directory for Kaggle:

```
mkdir .kaggle
```

Upload kaggle.json file and move the file to .kaggle directory:

```
mv kaggle.json .kaggle/
```

Install:
- -ZIP utilities
- -pip3 and virtual environment tools

```
sudo apt -y install zip
sudo apt -y install python3-pip python3.11-venv
```

Create python virtual environment

```
python3 -m venv pythondev
```

Change to pythondev directory
Activate the virtual environment

```
cd pythondev
source bin/activate
```

Install Kaggle cli tools

```
pip3 install kaggle
```

Download project dataset using the API command from Kaggle

```
kaggle datasets download -d kieranpoc/steam-reviews
```

Unzip the files

```
unzip steam-reviews.zip
```

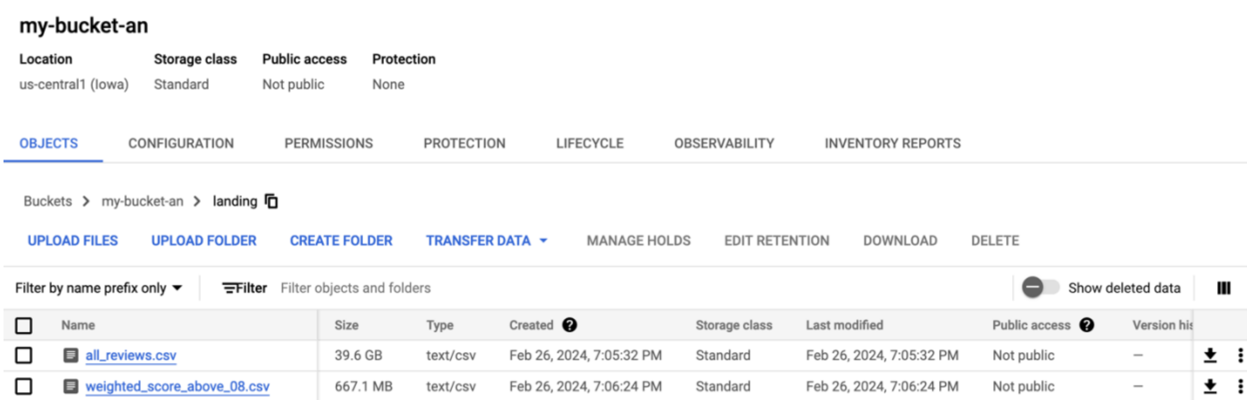
Create a bucket named "my-bucket-an" in the us-central1 region with my project id (healthy-genre-415522)

```
gcloud storage buckets create gs://my-bucket-an --
project=healthy-genre-415522 --default-storage-class=STANDARD --
location=us-central1 --uniform-bucket-level-access
```

Copy the .csv files from the local file system to the bucket created (my-bucket-an) landing folder.

```
gcloud storage cp all-reviews.csv gs://my-bucket-an/landing
gcloud storage cp weighted_score_above_08.csv gs://my-bucket-
an/landing
```

Results:



Exploratory Data Analysis (see Appendix C)

Create DataProc Cluster

```
gcloud dataproc clusters create cluster-d988 --enable-component-gateway --
region us-central1 --single-node --master-machine-type e2-standard-16 --
master-boot-disk-type pd-balanced --master-boot-disk-size 500 --image-version
2.2-debian12 --optional-components JUPYTER --max-idle 3600s --project
healthy-genre-415522
```

Import libraries needed.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Use nrows to do EDA in a few million rows.

```
skip = 0
total_rows_to_read = 113883717
nrows = 10000000
```

Loop to skip, read data in chunks, and do EDA.

```
skip = skip + nrows
while (skip < total_rows_to_read):
    print(f"Reading in {nrows} records starting at {skip}")
    reviews_df = pd.read_csv(f"{filepath}{filename}", sep=',',
skiprows=skip, nrows=nrows, header=None, names=column_names,
encoding='utf-8', on_bad_lines='skip')
    perform_EDA(reviews_df, filename)
    skip = skip + nrows
```

Number of records: 113,883,717

Data Columns: 24 variables

```
 #   Column                          Dtype
---  ------                          -----
 0   recommendationid                int64
 1   appid                           int64
 2   game                            object
 3   author_steamid                  int64
 4   author_num_games_owned          int64
 5   author_num_reviews              int64
 6   author_playtime_forever         float64
 7   author_playtime_last_two_weeks  float64
 8   author_playtime_at_review       int64
 9   author_last_played              float64
 10  language                        object
 11  review                          object
 12  timestamp_created               int64
 13  timestamp_updated               int64
 14  voted_up                        int64
 15  votes_up                        int64
 16  votes_funny                     int64
 17  weighted_vote_score             float64
 18  comment_count                   int64
 19  steam_purchase                  int64
 20  received_for_free               int64
 21  written_during_early_access     int64
 22  hidden_in_steam_china           int64
 23  steam_china_location            object
dtypes: float64(4), int64(16), object(4)
memory usage: 1.8+ GB
```

Missing Values:

```
all_reviews.csv Number of records: recommendationid
appid                             10000000
game                               9999577
author_steamid                    10000000
author_num_games_owned            10000000
author_num_reviews                10000000
author_playtime_forever           10000000
author_playtime_last_two_weeks    10000000
author_playtime_at_review         10000000
author_last_played                10000000
language                          10000000
review                             9999901
timestamp_created                 10000000
timestamp_updated                 10000000
voted_up                          10000000
votes_up                          10000000
votes_funny                       10000000
weighted_vote_score               10000000
comment_count                     10000000
steam_purchase                    10000000
received_for_free                 10000000
written_during_early_access       10000000
hidden_in_steam_china             10000000
steam_china_location                    32
dtype: int64
```

steam_china_location has a lot of missing values/records.

Fields containing null values:

```
all_reviews.csv Columns with null values
['game', 'author_playtime_forever', 'author_playtime_last_two_weeks', 'author_last_played', 'review', 'steam_china_
location']
```

Most common column with null values: ‘game’, ‘review’, ‘steam_china_location’

Summary statistics for numeric variables:

	recommendationid	appid	author_steamid	author_num_games_owned
count	1.000000e+07	1.000000e+07	1.000000e+07	1.000000e+07
mean	7.139602e+07	3.496837e+05	7.656120e+16	1.369616e+02
std	3.875101e+07	1.806161e+05	4.076164e+08	4.954795e+02
min	4.700000e+01	5.000000e+01	7.656120e+16	0.000000e+00
25%	3.797482e+07	4.131500e+05	7.656120e+16	0.000000e+00
50%	6.652163e+07	4.319600e+05	7.656120e+16	1.000000e+00
75%	1.022279e+08	4.571400e+05	7.656120e+16	1.090000e+02
max	1.494473e+08	5.042300e+05	7.656120e+16	3.335100e+04

	author_num_reviews	author_playtime_forever
count	1.000000e+07	9.999998e+06
mean	2.819634e+01	1.258051e+04
std	1.767524e+02	4.382571e+04
min	1.000000e+00	0.000000e+00
25%	2.000000e+00	5.150000e+02
50%	6.000000e+00	1.956000e+03
75%	1.800000e+01	8.006000e+03
max	1.044600e+04	5.440698e+06

	author_playtime_last_two_weeks	author_playtime_at_review
count	9.999998e+06	1.000000e+07
mean	5.276438e+01	6.303474e+03
std	4.663880e+02	2.367196e+04
min	0.000000e+00	0.000000e+00
25%	0.000000e+00	2.700000e+02
50%	0.000000e+00	8.520000e+02
75%	0.000000e+00	3.358000e+03
max	3.336900e+04	4.776595e+06

	votes_up	votes_funny	weighted_vote_score	comment_count
count	1.000000e+07	1.000000e+07	1.000000e+07	1.000000e+07
mean	2.153983e+00	1.082339e+05	1.817623e-01	1.324114e-01
std	3.125781e+01	2.156029e+07	2.513452e-01	1.875485e+00
min	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
50%	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
75%	1.000000e+00	0.000000e+00	4.927798e-01	0.000000e+00
max	2.962900e+04	4.294967e+09	9.966331e-01	2.515000e+03

	steam_purchase	received_for_free	written_during_early_access
count	1.000000e+07	1.000000e+07	1.000000e+07
mean	6.052688e-01	4.351700e-02	1.021650e-01
std	4.887929e-01	2.040178e-01	3.028652e-01
min	0.000000e+00	0.000000e+00	0.000000e+00
25%	0.000000e+00	0.000000e+00	0.000000e+00
50%	1.000000e+00	0.000000e+00	0.000000e+00
75%	1.000000e+00	0.000000e+00	0.000000e+00
max	1.000000e+00	1.000000e+00	1.000000e+00

	hidden_in_steam_china
count	1.000000e+07
mean	1.425258e-01
std	3.495886e-01
min	0.000000e+00
25%	0.000000e+00
50%	0.000000e+00
75%	0.000000e+00
max	1.000000e+00

Number of words in each review
(100000000-110000000)

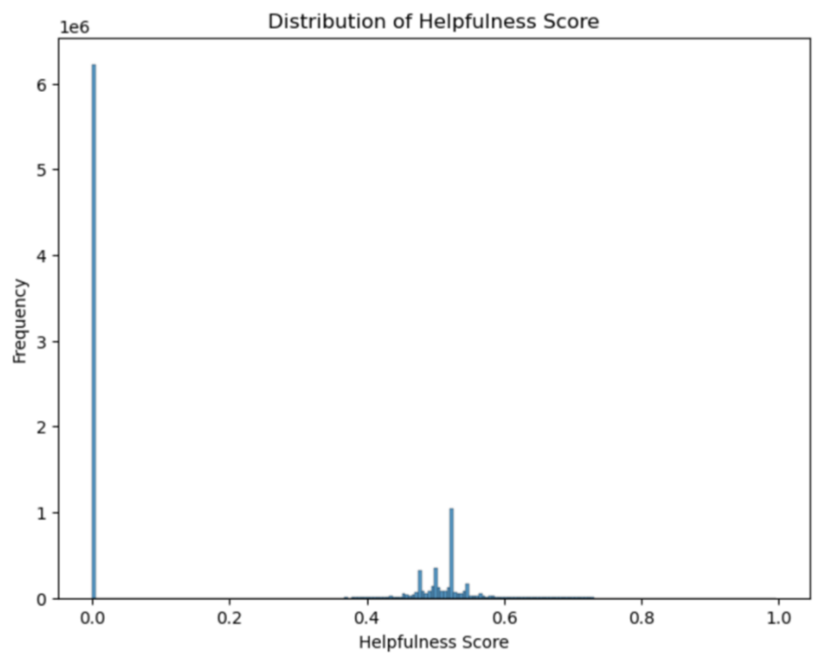
	review	num_words
0	HAOWAN	1
1	练枪首选哦	1
2	I miss her	3
3	rush b	2
4	its really good	3
...	...	...
9999995	Олды на месте!	3
9999996	This is the game.	4
9999997	Zajebista gra strzela sie do nazistów i nazist...	13
9999998	Mu bueno mu bueno	4
9999999	Great old-school shooter from my childhood! ...	85

[10000000 rows x 2 columns]

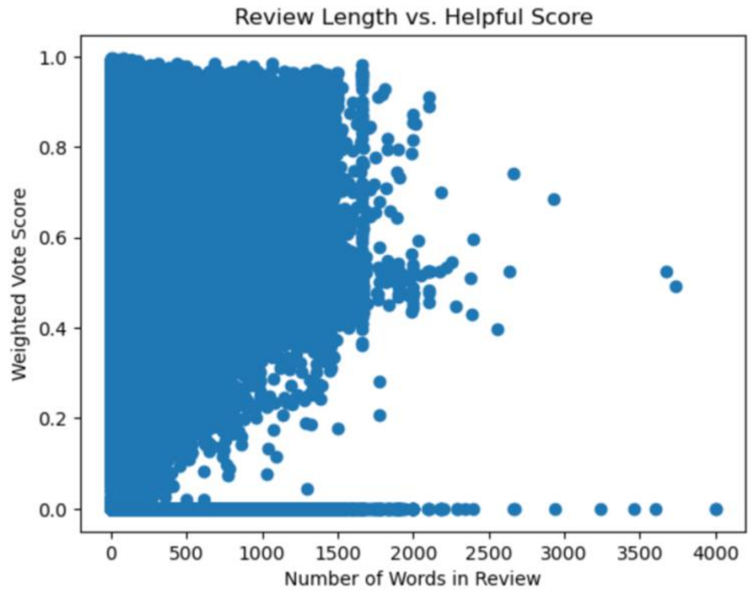
Number of characters in each review
(10000000- 20000000)

	review	num_characters
0	卑微铂金仔在线找人带我打派	13
1	输了可不好玩，所以我从来不玩🤖 咳咳，其实游戏是好游戏，要是服务器稳定点就更好了 —来自小...	57
2	ddd	3
3	不多bb，游戏是好游戏。服务器是真fw	19
4	Do not even bother trying to get into this gam...	408
...	...	...
9999995	Хорошая и интересная игра про шахматы. Можно к...	191
9999996	отличная игра спасибо разработчику!	35
9999997	Очень увлекательная и прикольная игрушка. Сам ...	219
9999998	Дуже кртуая игра 5 из 5!!!	26
9999999	Хорошие шахматы , есть режим против других игр...	151

[10000000 rows x 2 columns]



A lot of short reviews so most points are clustered together on the left side. As words increase, weighted vote score also increases.

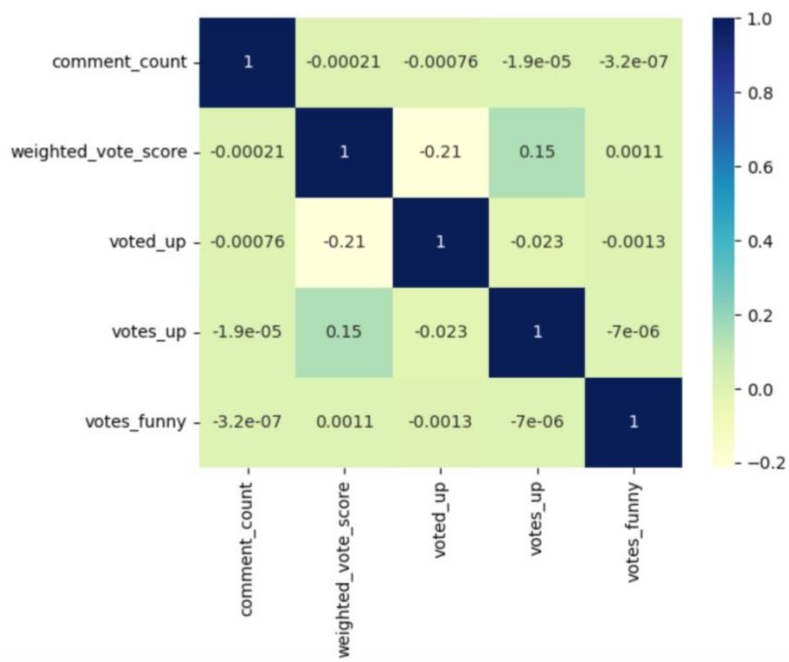


Correlation Matrix

weighted_vote_score (helpfulness score) has a correlation with votes up.

```
                    comment_count  weighted_vote_score  voted_up  votes_up  \
comment_count        1.000000e+00            -0.000214 -0.000763 -0.000019
weighted_vote_score -2.138862e-04             1.000000 -0.213476  0.145324
voted_up            -7.632242e-04            -0.213476  1.000000 -0.023371
votes_up            -1.916634e-05             0.145324 -0.023371  1.000000
votes_funny         -3.158265e-07             0.001063 -0.001282 -0.000007

                      votes_funny
comment_count       -3.158265e-07
weighted_vote_score  1.062881e-03
voted_up            -1.282338e-03
votes_up            -6.992989e-06
votes_funny          1.000000e+00
```



Cleaning Data

Read in 10000000 rows and drop column ‘steam_china_location’ because it has too many null values.

```
for filename in filename_list:
    print(f"Reading in {nrows} records starting at {skip}")
    df = pd.read_csv(f"{filepath}{filename}", sep=',',
skiprows=skip, nrows=nrows, header=None, names=column_names,
encoding='utf-8', on_bad_lines='skip')

    columns_to_drop = ['steam_china_location']

    df.drop(columns=columns_to_drop, inplace=True)
```

Remove rows with missing values.

```
df.dropna(inplace=True)
```

Create a copy of the dataframe.

```
df_first = df.copy()
```

Continue reading in 10000000 rows, drop column ‘steam_china_location’, and remove rows with missing values.

```
for filename in filename_list:
    print(f"Reading in {nrows} records starting at {skip}")
    df = pd.read_csv(f"{filepath}{filename}", sep=',',
skiprows=skip, nrows=nrows, header=None, names=column_names,
encoding='utf-8', on_bad_lines='skip')

    columns_to_drop = ['steam_china_location']

    df.drop(columns=columns_to_drop, inplace=True)

    print(df)
    skip = skip + nrows

df.dropna(inplace=True)

df_second = df.copy()
```

Combine df

```
combine_df = pd.concat([df_first, df_second, df_third,
df_fourth])
```

Drop columns not needed.

```
combine_df.drop(columns=['steam_purchase', 'received_for_free',
'written_during_early_access',
                   'hidden_in_steam_china', 'steam_purchase',
                   'timestamp_created', 'timestamp_updated',
'author_last_played',
                   'author_playtime_last_two_weeks',
'author_num_games_owned'], inplace=True)
```

```
combine_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 39995823 entries, 0 to 9999999
Data columns (total 14 columns):
 #   Column                       Dtype
---  ------                       -----
 0   recommendationid             int64
 1   appid                        int64
 2   game                         object
 3   author_steamid               int64
 4   author_num_reviews           int64
 5   author_playtime_forever      float64
 6   author_playtime_at_review    int64
 7   language                     object
 8   review                       object
 9   voted_up                     int64
 10  votes_up                     int64
 11  votes_funny                  int64
 12  weighted_vote_score          float64
 13  comment_count                int64
dtypes: float64(2), int64(9), object(3)
memory usage: 4.5+ GB
```

Write data to /cleaned folder as a Parquet file.

```
cleaned_filepath = "gcs://my-bucket-an/cleaned/"
```

```
combine_df.to_parquet(f"{cleaned_filepath}reviews1.parquet",
index=False)
```

Continue until end of data file from /landing.

Total of 4 files in /cleaned.

Feature Engineering and Modeling (see Appendix D)

Import libraries and files needed

```
from pyspark.ml.feature import Bucketizer, VectorAssembler
from pyspark.ml import Pipeline
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
sdf = spark.read.parquet("gs://my-bucket-
an/cleaned/reviews1.parquet")
```

```
sdf.columns
```

```
['recommendationid',
 'appid',
 'game',
 'author_steamid',
 'author_num_reviews',
 'author_playtime_forever',
 'author_playtime_at_review',
 'language',
 'review',
 'voted_up',
 'votes_up',
 'votes_funny',
 'weighted_vote_score',
 'comment_count']
```

- Author_playtime: spent more time playing a particular game might have deeper insights and experiences to share, = leading to more helpful reviews.
- Comment_count: number of comments might indicate its visibility and engagement level
- Votes_up & votes_funny: community feedback on the review, higher votes up and funny could imply that the review resonated with audience.

Model- predict helpfulness score of steam review using votes up, votes funny, comment count, number of author reviews, and author playtime.				
Features	Column	Data Type	Variable Type	
	votes_up	integer	numerical	bucketizer
	votes_funny	Integer	continuous	bucketizer
	comment_count	integer	continuous	bucketizer
	author_num_reviews	integer	continuous	bucketizer
	author_playtime_forever	integer	continuous	bucketizer
	author_playtime_at_review	integer	continuous	bucketizer
Label	weighted_vote_score	float	continuous	

Splits & bucketizer transformations for each column

```
splits_author_num_reviews = [0, 100, 500, 1000, 5000, 10500]
splits_playtime_forever = [0, 100, 500, 1000, 5000, 10000,
50000, 100000, 1000000, 10000000, float('inf')]

bucketizer_author_num_reviews =
Bucketizer(splits=splits_author_num_reviews,
inputCol="author_num_reviews",
outputCol="author_num_reviewsBucket")

bucketizer_playtime_forever =
Bucketizer(splits=splits_playtime_forever,
inputCol="author_playtime_forever",
outputCol="author_playtime_foreverBucket")

assembler =
VectorAssembler(inputCols=["author_num_reviewsBucket",

"author_playtime_foreverBucket",

"author_playtime_at_reviewBucket",
                                        "comment_countBucket",
                                        "votes_upBucket"],
                            outputCol="features")

# Transform the data using the pipeline
transformed_sdf = pipeline_model.transform(sdf)
```

```
Transformed features

+------------------+-----------------------+-------------------------+-------------+--------+-------------------+
|author_num_reviews|author_playtime_forever|author_playtime_at_review|comment_count|votes_up|features           |
+------------------+-----------------------+-------------------------+-------------+--------+-------------------+
|3                 |197.0                  |197                      |0            |0       |(5,[1],[1.0])      |
|21                |441.0                  |441                      |0            |0       |(5,[1],[1.0])      |
|1                 |1440.0                 |1313                     |0            |0       |(5,[1,2],[3.0,1.0])|
|4                 |1636.0                 |1612                     |0            |0       |(5,[1,2],[3.0,1.0])|
|2                 |197.0                  |197                      |0            |0       |(5,[1],[1.0])      |
|2                 |1685.0                 |1649                     |0            |0       |(5,[1,2],[3.0,1.0])|
|39                |11.0                   |11                       |0            |0       |(5,[],[])          |
|2                 |45119.0                |45119                    |0            |0       |(5,[1,2],[5.0,3.0])|
|4                 |1271.0                 |1202                     |0            |0       |(5,[1,2],[3.0,1.0])|
|60                |721.0                  |721                      |0            |0       |(5,[1],[2.0])      |
|1                 |12107.0                |12107                    |0            |0       |(5,[1,2],[5.0,3.0])|
|5                 |42519.0                |42515                    |0            |0       |(5,[1,2],[5.0,3.0])|
|3                 |6543.0                 |6322                     |0            |0       |(5,[1,2],[4.0,2.0])|
|1                 |25944.0                |25944                    |0            |0       |(5,[1,2],[5.0,3.0])|
|2                 |7818.0                 |7704                     |0            |0       |(5,[1,2],[4.0,2.0])|
|1                 |317.0                  |278                      |0            |0       |(5,[1],[1.0])      |
|1                 |89.0                   |77                       |0            |0       |(5,[],[])          |
|31                |141.0                  |115                      |0            |0       |(5,[1],[1.0])      |
|1                 |5684.0                 |5617                     |0            |0       |(5,[1,2],[4.0,2.0])|
|1                 |1762.0                 |1674                     |0            |0       |(5,[1,2],[3.0,1.0])|
+------------------+-----------------------+-------------------------+-------------+--------+-------------------+
only showing top 20 rows
```

Modeling

Split the data

```
trainingData, testData = transformed_sdf.randomSplit([0.7, 0.3],
seed=42)
```

Linear Regression Estimator & regression evaluator

```
linear_reg = LinearRegression(labelCol='weighted_vote_score')
evaluator = RegressionEvaluator(labelCol='weighted_vote_score',
metricName='rmse')
```

Train models & best model

```
all_models = cv.fit(trainingData)
bestModel = all_models.bestModel
test_results = bestModel.transform(testData)
```

Predicted weighted_vote_score (helpfulness score)

```
test_results.select('author_num_reviews',
'author_playtime_forever', 'comment_count', 'votes_up',
'weighted_vote_score', 'prediction').show(truncate=False)
```

author_num_reviews	author_playtime_forever	comment_count	votes_up	weighted_vote_score	prediction
92	76.0	0	0	0.0	0.19066581294500634
2	80062.0	0	0	0.0	0.15285773734772784
8	4056.0	0	0	0.0	0.14852783978391013
3	31927.0	0	1	0.528985500335693	0.1552867607201981
1	25901.0	0	0	0.0	0.1552867607201981
8	1664.0	0	2	0.54356849193573	0.14852783978391013
2	11617.0	0	0	0.0	0.14366979303896962
79	1177.0	0	1	0.523809552192688	0.1601448074651386
2	179711.0	0	0	0.0	0.1504287139752576
12	2391.0	0	0	0.0	0.1601448074651386
63	6704.0	0	0	0.476190477609634	0.15771578409266834
1	96879.0	0	1	0.523809552192688	0.15285773734772784
4	21128.0	0	0	0.0	0.1552867607201981
1	61382.0	0	0	0.0	0.14124076966649934
9	4510.0	0	0	0.0	0.1601448074651386
2	44704.0	0	0	0.0	0.1552867607201981
4	16456.0	0	0	0.0	0.1552867607201981
38	223.0	0	0	0.0	0.1766198218913076
5	418.0	0	0	0.0	0.1766198218913076
5	788.0	0	0	0.0	0.16257383083760887

RMSE & R2

```
rmse = evaluator.evaluate(test_results,
{evaluator.metricName:'rmse'})
r2 =evaluator.evaluate(test_results,{evaluator.metricName:'r2'})
print(f"RMSE: {rmse}  R-squared:{r2}")
```

RMSE: 0.23467789849773346   R-squared:0.12638481977611427

RMSE of 0.235 shows the average difference between average difference between the actual and predicted helpfulness scores. Lower value of RMSE represents a better model indicating that the model's predictions are closer to the actual values of the helpfulness score.

R squared value is 0.126 suggesting a weak fit and the model is not good at predicting scores.

Feature Engineering using MinMaxScaler

```
from pyspark.ml import Pipeline
from pyspark.ml.feature import MinMaxScaler, VectorAssembler

columnsA = [
    'author_num_reviews',
    'author_playtime_forever',
    'author_playtime_at_review',
    'comment_count',
    'votes_up',
    'votes_funny'
]
vector_assembler = VectorAssembler(inputCols=columnsA,
outputCol='features')
min_max_scaler = MinMaxScaler(inputCol="features",
outputCol="featuresA")

stages = [vector_assembler, min_max_scaler]
pipeline = Pipeline(stages=stages)
pipeline_model = pipeline.fit(sdf)
scaled_df = pipeline_model.transform(sdf)

scaled_df.select(['features', 'scaled_features']).show()
```

```
+--------------------+--------------------+
|            features|     scaled_features|
+--------------------+--------------------+
|[3.0,197.0,197.0,...|[1.91479176639540...|
|[21.0,441.0,441.0...|[0.00191479176639...|
|[1.0,1440.0,1313....|(6,[1,2],[2.46616...|
|[4.0,1636.0,1612....|[2.87218764959310...|
|[2.0,197.0,197.0,...|[9.57395883197702...|
|[2.0,1685.0,1649....|[9.57395883197702...|
|[39.0,11.0,11.0,0...|[0.00363810435615...|
|[2.0,45119.0,4511...|[9.57395883197702...|
|[4.0,1271.0,1202....|[2.87218764959310...|
|[60.0,721.0,721.0...|[0.00564863571086...|
|[1.0,12107.0,1210...|(6,[1,2],[0.00207...|
|[5.0,42519.0,4251...|[3.82958353279080...|
|[3.0,6543.0,6322....|[1.91479176639540...|
|[1.0,25944.0,2594...|(6,[1,2],[0.00444...|
|[2.0,7818.0,7704....|[9.57395883197702...|
|[1.0,317.0,278.0,...|(6,[1,2],[5.42899...|
|[1.0,89.0,77.0,0....|(6,[1,2],[1.52422...|
|[31.0,141.0,115.0...|[0.00287218764959...|
|[1.0,5684.0,5617....|(6,[1,2],[9.73451...|
|[1.0,1762.0,1674....|(6,[1,2],[3.01762...|
+--------------------+--------------------+
```

Modeling

author_num_reviews	author_playtime_forever	comment_count	votes_up	weighted_vote_score	prediction
92	76.0	0	0	0.0	0.1780857652836652
2	80062.0	0	0	0.0	0.1747724862304358
8	4056.0	0	0	0.0	0.17227446004569358
3	31927.0	0	1	0.528985500335693	0.17335627060513528
1	25901.0	0	0	0.0	0.1722582455314703
8	1664.0	0	2	0.54356849193573	0.1734062662875619
2	11617.0	0	0	0.0	0.17224745926339216
79	1177.0	0	1	0.523809552192688	0.17785401644833926
2	179711.0	0	0	0.0	0.1790208988584973
12	2391.0	0	0	0.0	0.17222343552049846
63	6704.0	0	0	0.476190477609634	0.17617056054666333
1	96879.0	0	1	0.523809552192688	0.1785632723541367
4	21128.0	0	0	0.0	0.17244679097786017
1	61382.0	0	0	0.0	0.1754405575306825
9	4510.0	0	0	0.0	0.17217268642594413
2	44704.0	0	0	0.0	0.17541180507946316
4	16456.0	0	0	0.0	0.17216826560637769
38	223.0	0	0	0.0	0.17407780317240504
5	418.0	0	0	0.0	0.1716316004824412
5	788.0	0	0	0.0	0.17164501588098693

RMSE & R2

```
RMSE: 0.24839715312907776 R-squared:0.0212562111632415
```

Feature Engineering using MinMaxScaler including review length

Importing Libraries

```
from pyspark.ml.feature import RegexTokenizer
from pyspark.ml import Pipeline
from pyspark.sql.functions import split, size
sdf = spark.read.parquet("gs://my-bucket-
an/cleaned/reviews1.parquet")
```

Filter only English reviews

```
sdf_en = sdf.filter(sdf['language'] == 'english')
```

Tokenize review text & count number of words

```
words_sdf = regexTokenizer.transform(sdf_en)
words_sdf_length = words_sdf.withColumn('num_words',
size(split(words_sdf['review'], ' ')))

assembler = VectorAssembler(inputCols=columnA + ['num_words'],
outputCol="featuresA")
```

Modeling

author_num_reviews	author_playtime_forever	comment_count	votes_up	num_words	weighted_vote_score	prediction
92	76.0	0	0	3	0.0	0.18600732066571432
2	80062.0	0	0	2	0.0	0.18404248355222713
8	4056.0	0	0	7	0.0	0.1821301674731944
3	31927.0	0	1	26	0.528985500335693	0.1831521177065865
8	1664.0	0	2	3	0.54356849193573	0.18318439200677247
2	11617.0	0	0	2	0.0	0.18211079904827582
1	124510.0	0	0	11	0.0	0.18505448787168402
2	179711.0	0	0	5	0.0	0.18710450087626648
63	6704.0	0	0	24	0.476190477609634	0.18477222791304382
2	47344.0	0	1	4	0.523809552192688	0.1835245890643009
5	10771.0	0	0	1	0.0	0.18205795464151298
104	14.0	0	0	83	0.0	0.18659330621395245
2	44704.0	0	0	5	0.0	0.18389269729496902
15	282452.0	0	0	4	0.0	0.1902607079111116
17	2965.0	0	0	20	0.0	0.18246679749479383

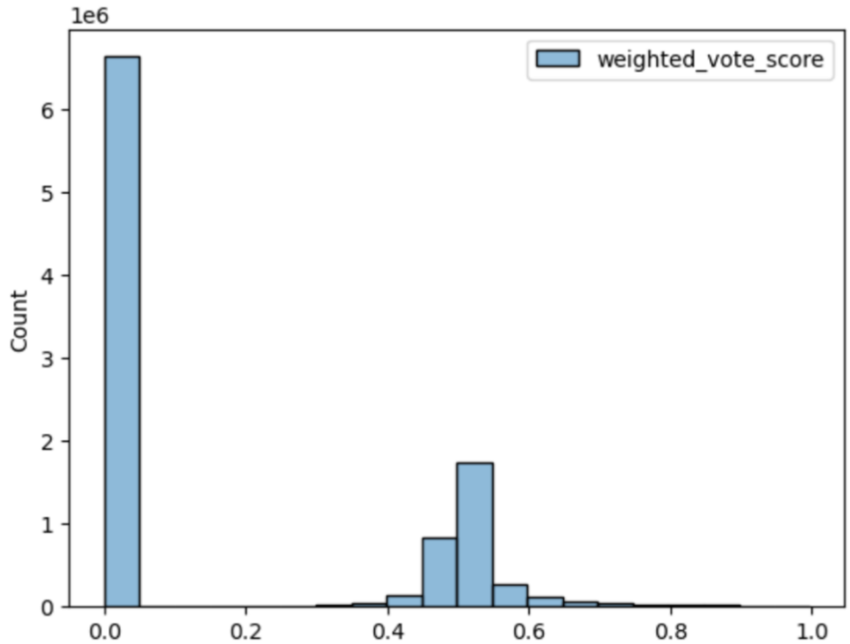
RMSE & R2

```
RMSE: 0.24885898660858216 R-squared:0.018993799809560397
```

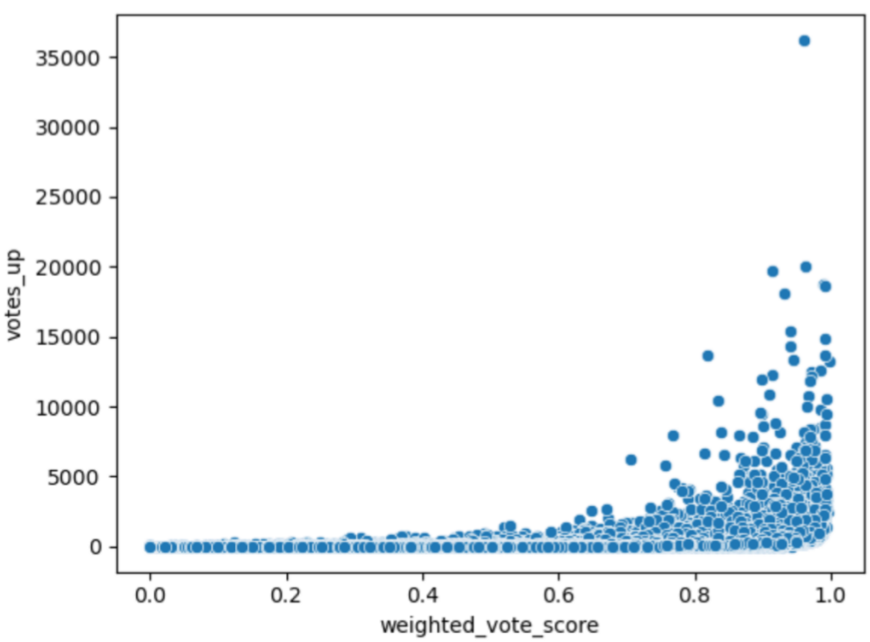
Using bucketizer, the model gave the lowest RMSE indicating better performance in terms of predicting the helpfulness score.
Using minmaxscaler, the model gave highest r-squared value.

Data Visualizing (see Appendix E)

Distribution of weighted_vote_score

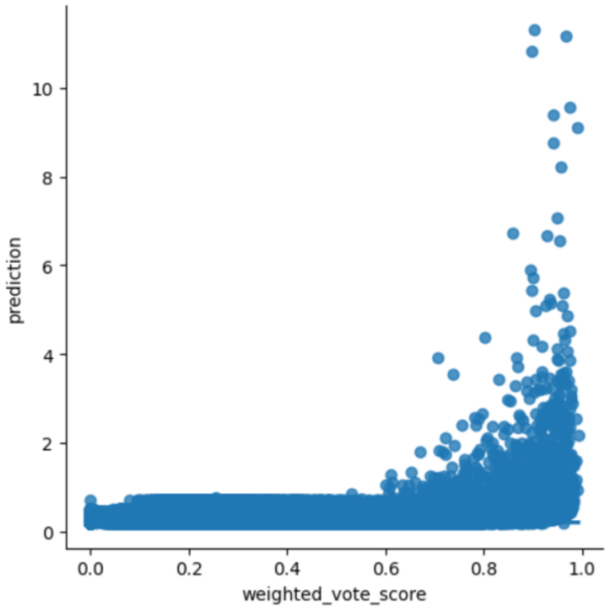


Scatterplot

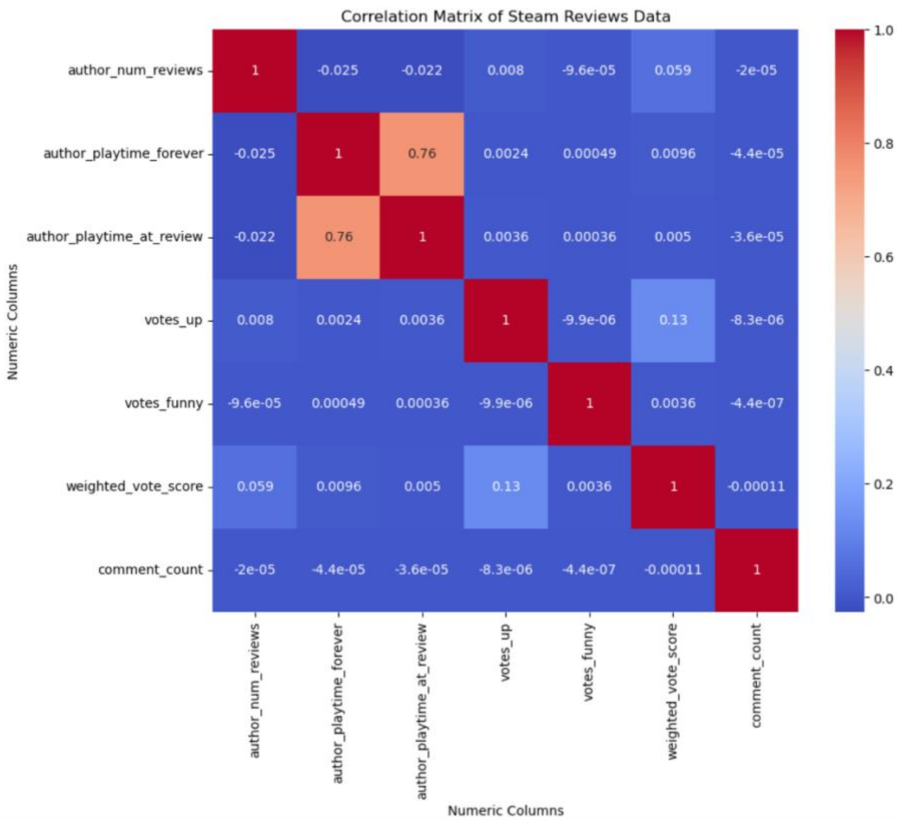


Visualizes the relationship between the votes up a review has and the given helpfulness score of the review. The helpfulness score increases as there are more up votes a review has.

Regression results



Correlation Matrix



From the correlation matrix, votes up has the highest correlation between weighted vote score, followed by the author's number of reviews. Comment count has the most negative relationships.

Summary & Conclusions

This project aims to predict the helpfulness score of Steam reviews, represented by the 'weighted_vote_score', using various review-related features. The data processing pipeline involves cleaning the data, feature engineering, and building a predictive model using linear regression. The project's final goal is to evaluate the model's performance and understand the key features influencing the helpfulness scores.

Data Processing Pipeline

1. Loading the data
   Parquet file stored in Google Cloud Storage.
2. Feature Engineering
   Bucketizer & MinMaxScaler on features:
      'author_num_reviews'
      'author_playtime_forever'
      'author_playtime_at_review'
      'comment_count'
      'votes_up'
      'votes_funny'
   Tokenizer on:
      'review'
   VectorAssembler to combine them into a single feature vector
3. Modeling & Evaluation
   The transformed data is split into training and test sets. The training set is used to train the model, while the test set is used to evaluate the model's performance.
   Create linear regression model using 'weighted_vote_score'
   Perform cross validation and select the best model
   Evaluate the model using RegressionEvaluator to calculate the RMSE and R-squared

This processes the data, transforms features, trains a linear regression model, and evaluates its performance. The main challenge was determining the appropriate feature transformations and ensuring that all features were bucketized.

The RMSE and R-squared of the three models used indicate that the model has moderate level of error, but it does not capture most of the factors influencing the helpfulness score. This suggests the need for different models or additional features to improve predictive performance.
Based on the coefficients, 'author_num_reviews', 'reviews', and 'voted_up', were identified as the most influential features in predicting the helpfulness score.

Github url: https://github.com/amyyning/cis4130-project

Appendix B

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

def perform_EDA(df : pd.DataFrame, filename : str):

    print(f"{filename} Number of records: {df.count()}" )

    print(f"{filename} Number of duplicate records: { len(df)-
len(df.drop_duplicates())}" )

    print(f"{filename} Info")
    print(df.info())

    print(f"{filename} Columns with null values")

print(reviews_df.columns[reviews_df.isnull().any()].tolist())
    rows_with_null_values =
reviews_df.isnull().any(axis=1).sum()
    print(f"{filename} Number of Rows with null values:
{rows_with_null_values}" )

    numeric_summary = df.describe()
    print("Summary statistics for numeric variables:")
    print(numeric_summary)

filepath = "gcs://my-bucket-an/landing/"
filename_list = ['all_reviews.csv']

skip = 0
total_rows_to_read = 113883717
nrows = 10000000

for filename in filename_list:
    print(f"Working on file: {filename}")
    reviews_df = pd.read_csv(f"{filepath}{filename}", sep=',',
skiprows=skip, nrows=nrows, encoding='utf-8',
on_bad_lines='skip')

    perform_EDA(reviews_df,filename)

column_names = reviews_df.columns

skip = skip + nrows
```

```
while (skip < total_rows_to_read):
    print(f"Reading in {nrows} records starting at {skip}")
    reviews_df = pd.read_csv(f"{filepath}{filename}", sep=',',
skiprows=skip, nrows=nrows, header=None, names=column_names,
encoding='utf-8', on_bad_lines='skip')

    perform_EDA(reviews_df, filename)
    # Increment the skip
    skip = skip + nrows

# Number of words in each review
reviews_df['num_words'] = reviews_df['review'].apply(lambda x:
len(str(x).split()))
# Print number of words for each review
print(reviews_df[['review', 'num_words']])

# Characters in each review
reviews_df['num_characters'] = reviews_df['review'].apply(lambda
x: len(str(x)))
print(reviews_df[['review', 'num_characters']])


# Distribution of Helpfulness Score
sns.histplot(reviews_df['weighted_vote_score'])
plt.title("Distribution of Helpfulness Score")
plt.xlabel("Helpfulness Score")
plt.ylabel("Frequency")
plt.show()

lp = sns.lmplot(x='num_words', y='weighted_vote_score',
data=reviews_df)
lp.set_axis_labels("Number of Words in Review", "Weighted Vote
Score")
plt.show()


numeric_columns =
["comment_count","weighted_vote_score","voted_up","votes_up","vo
tes_funny"]
df = reviews_df[ numeric_columns ]
print(df.corr())
dataplot = sns.heatmap(df.corr(), cmap="YlGnBu", annot=True)
```

Appendix C

```
import pandas as pd

# Define file path
filepath = "gcs://my-bucket-an/landing/"
filename_list = ['all_reviews.csv']

total_rows_to_read = 113883717
nrows = 10000000
skip = 0

for filename in filename_list:
    df = pd.read_csv(f"{filepath}{filename}", sep=',',
skiprows=skip, nrows=nrows, encoding='utf-8',
on_bad_lines='skip')

    column_names = df.columns

    # Define columns to drop
    columns_to_drop = ['steam_china_location']

    df.drop(columns=columns_to_drop, inplace=True)

    print(df)

df.dropna(inplace=True)
df_first = df.copy()

skip = 10000000
for filename in filename_list:
    print(f"Reading in {nrows} records starting at {skip}")
    df = pd.read_csv(f"{filepath}{filename}", sep=',',
skiprows=skip, nrows=nrows, header=None, names=column_names,
encoding='utf-8', on_bad_lines='skip')

    columns_to_drop = ['steam_china_location']

    df.drop(columns=columns_to_drop, inplace=True)

    print(df)
    skip = skip + nrows

df.dropna(inplace=True)
df_second = df.copy()
```

```
combine_df = pd.concat([df_first, df_second, df_third,
df_fourth])

combine_df.drop(columns=['steam_purchase', 'received_for_free',
'written_during_early_access',
                   'hidden_in_steam_china', 'steam_purchase',
                   'timestamp_created', 'timestamp_updated',
'author_last_played',
                   'author_playtime_last_two_weeks',
'author_num_games_owned'], inplace=True)

cleaned_filepath = "gcs://my-bucket-an/cleaned/"

combine_df.to_parquet(f"{cleaned_filepath}reviews1.parquet",
index=False)

combine_df = pd.concat([df_fifth, df_sixth, df_seventh,
df_eighth])

combine_df.to_parquet(f"{cleaned_filepath}reviews2.parquet",
index=False)
```

Appendix D

```
from pyspark.ml.feature import Bucketizer, VectorAssembler
from pyspark.ml import Pipeline
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

sdf = spark.read.parquet("gs://my-bucket-
an/cleaned/reviews1.parquet")

# splits
splits_author_num_reviews = [0, 100, 500, 1000, 5000, 10500]
splits_playtime_forever = [0, 100, 500, 1000, 5000, 10000,
50000, 100000, 1000000, 10000000, float('inf')]
splits_playtime_at_review = [0, 1000, 5000, 10000, 50000,
100000, 500000, 1000000, 5000000, 10000000, 50000000, 100000000,
500000000, 1000000000, 5000000000]
splits_comment_count = [0, 10, 20, 50, 100, 200, 300,
float("inf")]
splits_votes_up = [0, 10, 50, 100, 200, 500, 1000, 2000, 5000,
10000, 20000, 30000, 40000, 50000, 60000, float('inf')]

# bucketizer
bucketizer_author_num_reviews =
Bucketizer(splits=splits_author_num_reviews,
inputCol="author_num_reviews",
outputCol="author_num_reviewsBucket")

bucketizer_playtime_forever =
Bucketizer(splits=splits_playtime_forever,
inputCol="author_playtime_forever",
outputCol="author_playtime_foreverBucket")

bucketizer_playtime_at_review =
Bucketizer(splits=splits_playtime_at_review,
inputCol="author_playtime_at_review",
outputCol="author_playtime_at_reviewBucket")

bucketizer_comment_count =
Bucketizer(splits=splits_comment_count,
inputCol="comment_count", outputCol="comment_countBucket")

bucketizer_votes_up = Bucketizer(splits=splits_votes_up,
inputCol="votes_up", outputCol="votes_upBucket")
```

```
assembler =
VectorAssembler(inputCols=["author_num_reviewsBucket",

"author_playtime_foreverBucket",

"author_playtime_at_reviewBucket",
                                   "comment_countBucket",
                                       "votes_upBucket"],
                            outputCol="features")

r_pipeline= Pipeline(stages=[bucketizer_author_num_reviews,
                            bucketizer_playtime_forever,
                            bucketizer_playtime_at_review,
                            bucketizer_comment_count,
                            bucketizer_votes_up,
                            assembler])

pipeline_model = r_pipeline.fit(sdf)

# Pipeline
transformed_sdf = pipeline_model.transform(sdf)

print("Transformed features")
transformed_sdf.select("author_num_reviews",
                       "author_playtime_forever",
                       "author_playtime_at_review",
                       "comment_count",
                       "votes_up",
                       "features").show(truncate=False)

transformed_sdf.write.parquet("gs://my-bucket-
an/trusted/reviews_with_features.parquet")

# Split data into training and test sets
trainingData, testData = transformed_sdf.randomSplit([0.7, 0.3],
seed=42)

# Linear Regression Estimator
linear_reg = LinearRegression(labelCol='weighted_vote_score')

# Create regression evaluator
evaluator = RegressionEvaluator(labelCol='weighted_vote_score',
metricName='rmse')

stagess = [linear_reg]
```

```python
regression_pipe = Pipeline(stages=stagess)

# Create a grid to hold hyperparameters
grid = ParamGridBuilder()
# Build the parameter grid
grid = grid.build()

# Create the CrossValidator using the hyperparameter grid
cv = CrossValidator(estimator=regression_pipe,
                    estimatorParamMaps=grid,
                    evaluator=evaluator,
                    numFolds=3)

# Train the models
all_models = cv.fit(trainingData)

bestModel = all_models.bestModel

# Use 'bestModel' to predict the test set
test_results = bestModel.transform(testData)

# Show the predicted weighted_vote_score
test_results.select('author_num_reviews',
'author_playtime_forever', 'comment_count', 'votes_up',
'weighted_vote_score', 'prediction').show(truncate=False)

# Calculate RMSE % R2
rmse = evaluator.evaluate(test_results,
{evaluator.metricName:'rmse'})
r2 =evaluator.evaluate(test_results,{evaluator.metricName:'r2'})
print(f"RMSE: {rmse}  R-squared:{r2}")

bestModel.write().save("gs://my-bucket-an/models/model1")

for i in range(len(coefficients)):
    print(testData.columns[i], coefficients[i])

feature_importance = sorted(list(zip(testData.columns[:-1],
map(abs, coefficients))), key=lambda x: x[1], reverse=True)

print("Feature Importance:")
for feature, importance in feature_importance:
    print("  {}: {:.3f}".format(feature, importance))
```

MinMaxScaler

```
from pyspark.ml.feature import MinMaxScaler, VectorAssembler

columns_a = [
    'author_num_reviews',
    'author_playtime_forever',
    'author_playtime_at_review',
    'comment_count',
    'votes_up',
    'votes_funny'
]
vector_assembler = VectorAssembler(inputCols=columns_a,
outputCol='features')

min_max_scaler = MinMaxScaler(inputCol="features",
outputCol="scaled_features")

stages = [vector_assembler, min_max_scaler]

pipeline = Pipeline(stages=stages)
pipeline_model = pipeline.fit(sdf)
scaled_df = pipeline_model.transform(sdf)

scaled_df.show()

trainingData, testData = scaled_df.randomSplit([0.7, 0.3],
seed=42)

linear_reg = LinearRegression(labelCol='weighted_vote_score')
evaluator = RegressionEvaluator(labelCol='weighted_vote_score',
metricName='rmse')


stagess = [linear_reg]
regression_pipe = Pipeline(stages=stagess)
grid = ParamGridBuilder()
grid = grid.build()

cv = CrossValidator(estimator=regression_pipe,
                    estimatorParamMaps=grid,
                    evaluator=evaluator,
                    numFolds=3)

all_models = cv.fit(trainingData)
bestModel = all_models.bestModel
```

```
test_results = bestModel.transform(testData)

test_results.select('author_num_reviews',
'author_playtime_forever', 'comment_count', 'votes_up',
'weighted_vote_score', 'prediction').show(truncate=False)

bestModel.write().save("gs://my-bucket-an/models/model2")
```

MinMaxScaler with review length

```
from pyspark.ml import Pipeline
from pyspark.ml.feature import MinMaxScaler, VectorAssembler,
RegexTokenizer
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
from pyspark.sql.functions import split, size


sdf = spark.read.parquet("gs://my-bucket-
an/cleaned/reviews1.parquet")
sdf_en = sdf.filter(sdf['language'] == 'english')

columns_a = [
    'author_num_reviews',
    'author_playtime_forever',
    'author_playtime_at_review',
    'comment_count',
    'votes_up',
    'votes_funny'
]

vector_assembler = VectorAssembler(inputCols=columns_a,
outputCol='features')
min_max_scaler = MinMaxScaler(inputCol="features",
outputCol="scaled_features")

regexTokenizer = RegexTokenizer(inputCol="review",
outputCol="words", pattern="\\w+", gaps=False)

words_sdf = regexTokenizer.transform(sdf_en)

words_sdf_length = words_sdf.withColumn('num_words',
size(split(words_sdf['review'], ' ')))

assembler = VectorAssembler(inputCols=columns_a + ['num_words'],
```

```
outputCol="featuresA")

stages = [vector_assembler, min_max_scaler, assembler]
pipeline = Pipeline(stages=stages)
pipeline_model = pipeline.fit(words_sdf_length)
transformed_df = pipeline_model.transform(words_sdf_length)

trainingData, testData = transformed_df.randomSplit([0.7, 0.3],
seed=42)

linear_reg = LinearRegression(labelCol='weighted_vote_score')
evaluator = RegressionEvaluator(labelCol='weighted_vote_score',
metricName='rmse')

stagess = [linear_reg]
regression_pipeline = Pipeline(stages=stagess)

grid = ParamGridBuilder()
grid = grid.build()

cv = CrossValidator(estimator=regression_pipe,
                    estimatorParamMaps=grid,
                    evaluator=evaluator,
                    numFolds=3)

all_models = cv.fit(trainingData)

bestModel = all_models.bestModel
test_results = bestModel.transform(testData)

test_results.select('author_num_reviews',
'author_playtime_forever', 'comment_count', 'votes_up',
'num_words', 'weighted_vote_score',
'prediction').show(truncate=False)

bestModel.write().save("gs://my-bucket-an/models/model3")
```

Appendix E

```
import seaborn as sns
import matplotlib.pyplot as plt
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.stat import Correlation
import pandas as pd

#scatterplot
df2 = sdf.select('votes_up','weighted_vote_score').sample(False,
0.25).toPandas()
sns.scatterplot(x='weighted_vote_score', y='votes_up', data=df2)
plt.show()

#correlation matrix
vector_column = "correlation_features"
numeric_columns = ['author_num_reviews',
'author_playtime_forever', 'author_playtime_at_review',
'votes_up', 'votes_funny', 'weighted_vote_score',
'comment_count']
assembler = VectorAssembler(inputCols=numeric_columns,
outputCol=vector_column)
sdf_vector = assembler.transform(sdf).select(vector_column)

matrix = Correlation.corr(sdf_vector,
vector_column).collect()[0][0]
correlation_matrix = matrix.toArray().tolist()

correlation_matrix_df = pd.DataFrame(data=correlation_matrix,
columns=numeric_columns, index=numeric_columns)

plt.figure(figsize=(10, 8))
hm = sns.heatmap(correlation_matrix_df, annot=True,
cmap="coolwarm")
plt.title('Correlation Matrix of Steam Reviews Data')
plt.xlabel('Numeric Columns')
plt.ylabel('Numeric Columns')
plt.show()

#relationship plot
df = test_results.select('weighted_vote_score',
'prediction').toPandas()
sns.lmplot(x='weighted_vote_score', y='prediction', data=df)

#distribution of helpfulness score
```

```
df3 = sdf.select('weighted_vote_score').sample(False,
0.25).toPandas()
sns.histplot(df3, bins=20)
```