Rashedul Kabir

Milestone #1

Dataset: 100 Million+ Steam Reviews
URL: https://www.kaggle.com/datasets/kieranpoc/steam-reviews

- author steamid
- number of games owned by author
- number of reviews by author
- playtime all time by author
- playtime over the last 2 weeks by author
- playtime at the time of the review by author
- when they last played the game
- language
- time created
- time updated
- number of people who voted the review up
- number of people who voted the review funny
- 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

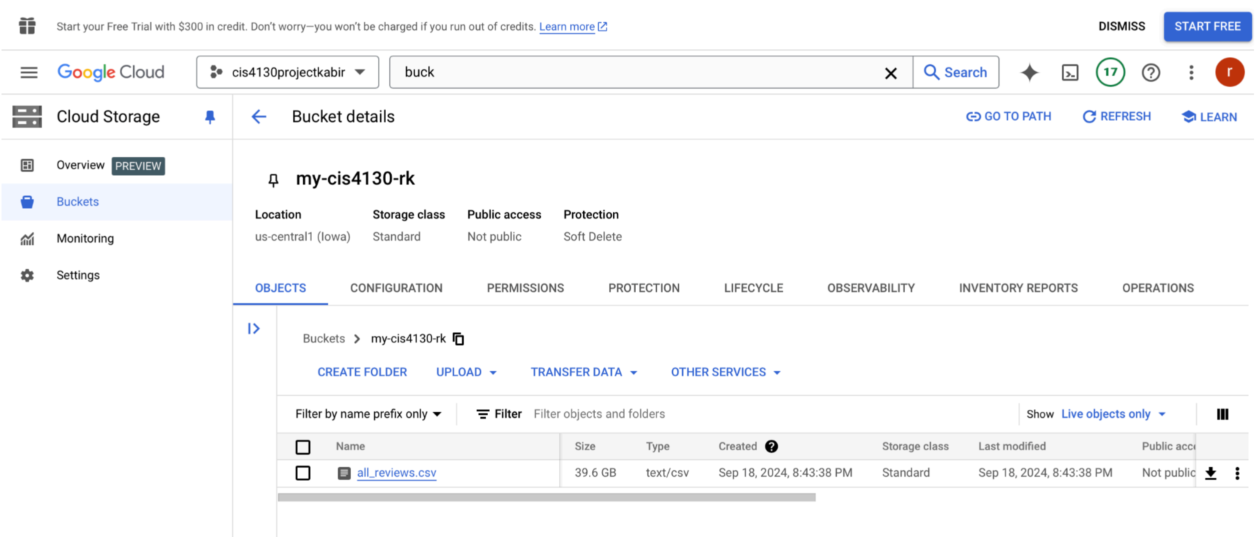
Description: This dataset contains review data for various games that are on the Steam platform.

Use Cases:

- I will then be implementing a logistic regression model predicting whether the weighted_vote_score is greater than .4

Milestone #2

Summary: I copied the api for the Steam reviews dataset and pasted it into the VM, which downloaded the file. I then unzipped the file. Then I created the bucket with the name my-cis-4130-rk, then moved the file into that bucket.

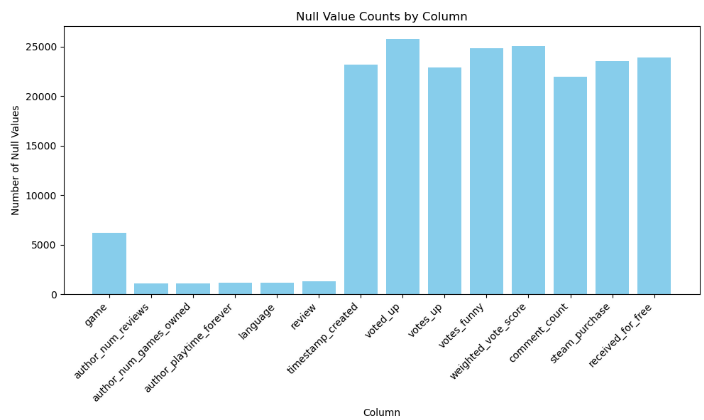


Milestone #3

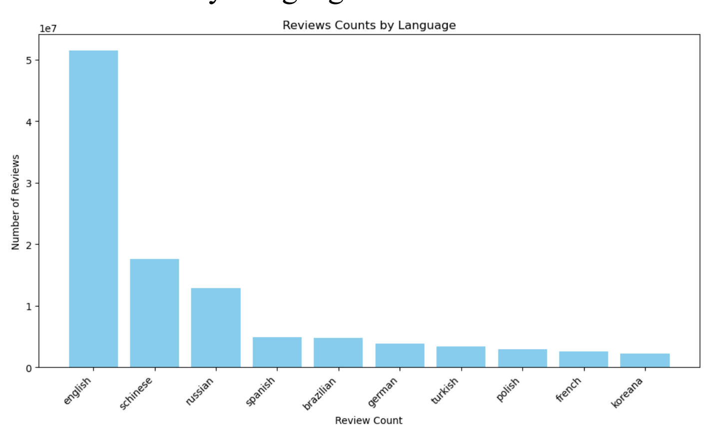
Exploratory Data Analysis:

Count of rows: 113885601

Count of nulls:



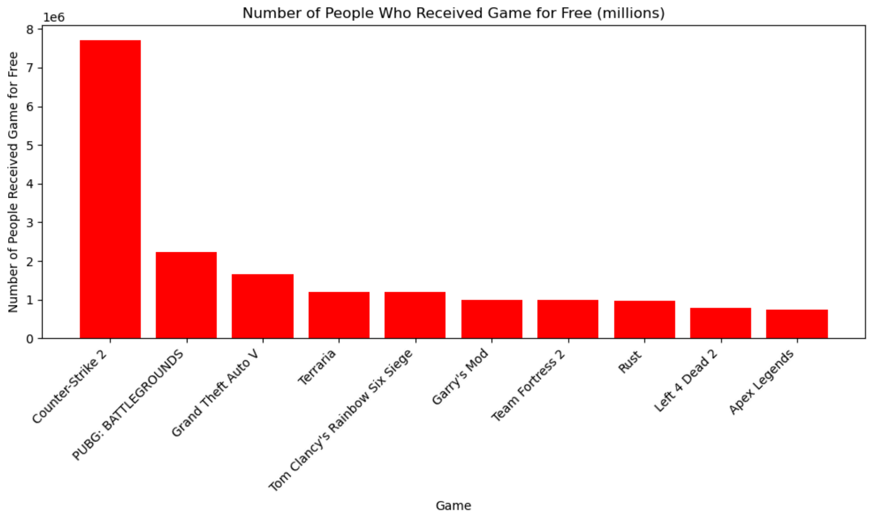
Review Count by Language:



Oldest Review Timestamp: 1969-12-31 23:59:55

Latest Review Timestamp: 2286-11-20 17:46:40

Number of People who Received Game for Free:



Review Word Count:

```
+----------+--------+
|      word|   count|
+----------+--------+
|          |83342328|
|       the|66241043|
|         a|47649932|
|       and|46604543|
|        to|43477156|
|      game|33954497|
|        of|31345267|
|        is|31177617|
|         I|28076022|
|       you|23179773|
|        it|21737227|
|      this|19893804|
|        in|18166429|
|       for|16144114|
|       but|14009503|
|      that|13999385|
|      with|13621325|
|EnergyDess|12308080|
|        de|11779498|
|        on|10403401|
+----------+--------+
```

:

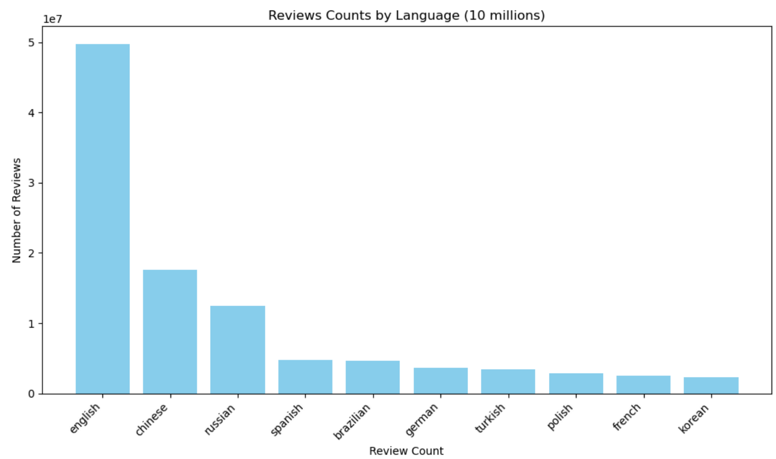
Conclusion: Majority of the reviews are in english, which I will most likely filter into a new df and work with, it will still be about half of the data which is around 20GB. I visualize various aspects of the data to see how it may affect model performance, for example, I visualized the number of people who received a particular game for free, this may affect their review. Also I filtered by the max & min timestamp which showed some errors with the value, so I will try to address it in the cleaning portion. I will also fix the misspelling of the language that the reviews are in.

Cleaning Data:

Timestamp_created Change:

```
+---------+------------------+----------------------+-----------------------+--------+-----------------------------------+---
---------------+--------+--------+-----------+-------------------+-------------+--------------+------------------+----+
|     game|author_num_reviews|author_num_games_owned|author_playtime_forever|language|                             review|  t
imestamp_created|voted_up|votes_up|votes_funny|weighted_vote_score|comment_count|steam_purchase|received_for_free|year|
+---------+------------------+----------------------+-----------------------+--------+-----------------------------------+---
---------------+--------+--------+-----------+-------------------+-------------+--------------+------------------+----+
|Stellaris|                 5|                   318|                  34545| english|               Stelleris is one ...|201
8-07-22 11:50:50|       1|       0|          0|                0.0|            0|             1|                 0|2018|
|Stellaris|                 4|                     0|                  21964| russian|               Нормальная игра, ...|201
8-07-22 11:28:04|       1|       3|          0|   0.56692910194397|            0|             1|                 0|2018|
|Stellaris|                21|                     0|                   2467|  korean|실시간이지만 턴제게임과 같은 엄...|2018-07-22
11:09:22|       1|       0|          0|                0.0|            0|             1|                 0|2018|
|Stellaris|                19|                     0|                  68206| english|               One of the all ti...|201
8-07-22 10:04:48|       1|       2|          1|  0.515306115150452|            0|             1|                 0|2018|
|Stellaris|                 3|                     0|                  83452| english|               Stellaris is an a...|201
8-07-22 09:49:39|       1|       0|          0|                0.0|            0|             1|                 0|2018|
+---------+------------------+----------------------+-----------------------+--------+-----------------------------------+---
---------------+--------+--------+-----------+-------------------+-------------+--------------+------------------+----+
only showing top 5 rows
```

Language Changes:



Timestamp Filtering:

min_timestamp	max_timestamp
1969-12-31 23:59:55	2023-11-03 16:16:25

Conclusion: I saw the timestamp column was not in the right format so I decided to change that first. In the EDA step I noticed when getting the max year, it was 2068, which is most likely an error. So I decided to filter out any reviews that were after 2024. In the EDA step I also noticed some languages were misspelled, so I decided to address that. Then I dropped all nans as I noticed it would only drop about 10,000 rows or so, which would not impact my data very much.

Milestone #4

Column Name	Data Type	Feature Engineering
game	String	Indexer then one hot encoder
author_num_reviews	Continuous	Standardized scaler
author_num_games_owned	Continuous	Standardized scaler
author_playtime_forever	Continuous (minutes)	
review	string	Tokenize->hashing->IDF
voted_up	binary	
votes_up	Continuous	Standardized scaler
votes_funny	binary	Standardized scaler
weighted_vote_score	continuous	Binary encoding. Score .70<= is a good score(1). <.70 is a bad score(0)
comment_count	continuous	Standardized scaler
steam_purchase	binary	
received_for_free	binary	

After reading in the data I dropped columns which I did not think would be of much use in the model. After which I filtered out nulls and created the label that I was going to predict which was the binary comment score. This column consisted of 1 which meant that the comment had a weighted vote score of less than or equal to .4 and 0 if it was less than that. I did this to handle the class imbalances. Next I used the csv file with the tip 200 games and filtered my data by it. After that I filtered out reviews which were not in english. I then cleaned my data some more as when I created my pipeline I continuously got errors. So I filtered out rows of columns which contained letters, when it is supposed to be a binary value. Moreover, I filled any nans with 0 and dropped some to make sure.

I then created the pipeline, with the first step being to index and encode the game column. I applied a string indexer and then a one hot encoder into a vector. Next I tokenized the review column and used Hashing and IDF library to encode it into a vector. Then I used the vector assembler on the numeric columns which I wanted to scale. These were turned into a single column. Then the features were all assembled into a final feature vector, with the column name "final_features". I then applied the pipeline on the entire dataframe and saved the processed dataframe.

Then I applied the Logistic Regression and evaluated the predictions. I received an accuracy of .65 which is okay for the model. I then sampled half of the data frame to get hyperparameters. I used the hyperparameter grid and cross validator. The results were:

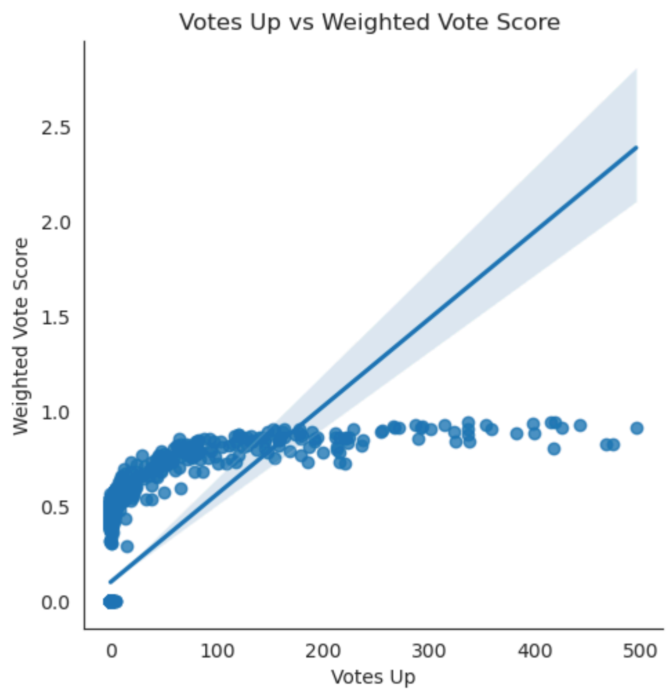
Best regParam: 0.01
Best elasticNetParam: 0.0
Best maxIter: 50

I then used these hyperparameters and trained my model again however the model performed the same with an accuracy of .65. I saved this model to my google storage

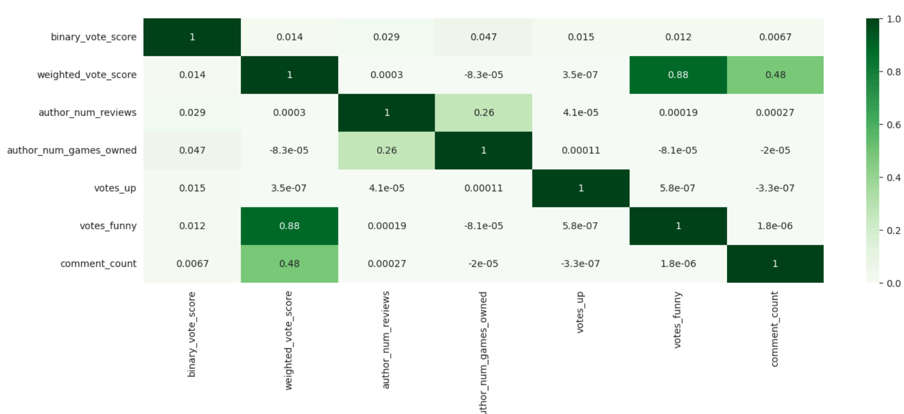
I decided to try to find ways to increase the accuracy, so I listed the coefficients. The columns with the lowest were the steam_purchased column and scaled features. So it may be better to drop features for future model improvement.

Milestone #5

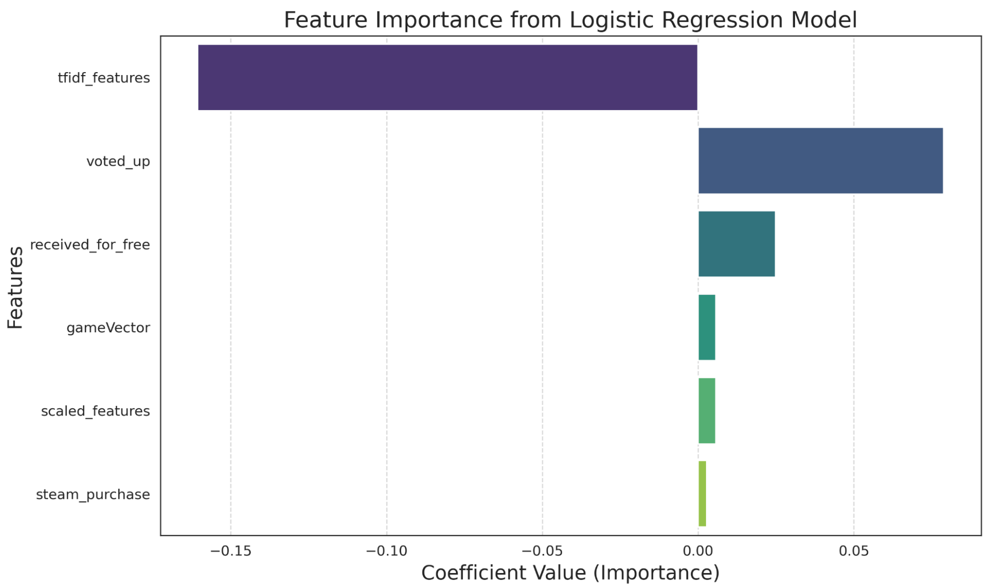
The first visualization I decided to make is a relationship plot between votes up and weighted score. I did this just out of curiosity to see if a trend could be identified. It does not look like there is much of a trend as when votes_up increases, the weighted vote score does not follow that trend.



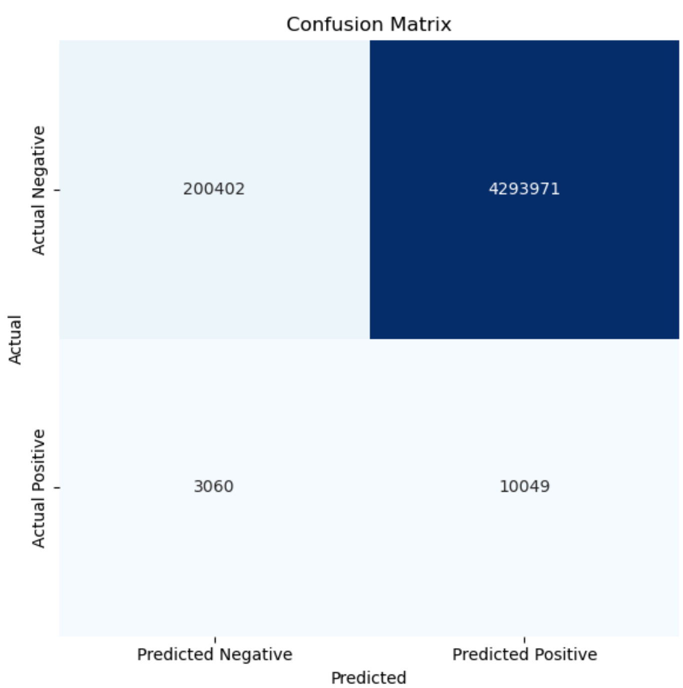
The next visualization is this correlation matrix to view which feature is correlated best to the target variable.



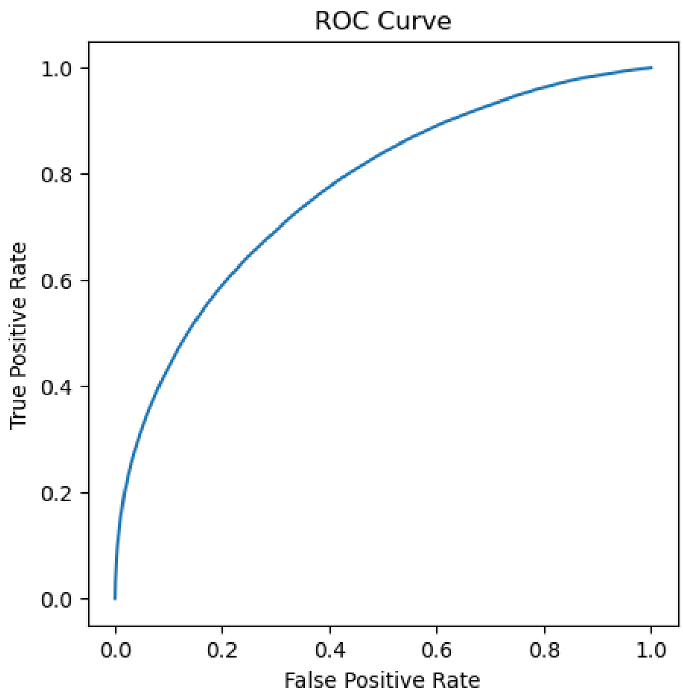
The next plot I created after running the model is visualizing the importance of features. This is to drop features which may not be important so that the model can be improved in the future.



The next visualization is a confusion matrix of the results of running the model. I ran the model on the entire dataset, and it did alright not the best, if we were to compare the predictions. .



The last visualization is the ROC curve which shows the relationship of tests true positive rates and false positive rates.



Milestone #6

Github Repository Link: https://github.com/rashedulkabir730/SteamReviewsETL-ML

Project Summary:

This project presents a data pipeline & ML model for the Steam Reviews data set. The ETL pipeline utilized GCP & Spark for transformations and modeling building. The goal of the model was to predict whether a review would have a weighted score of greater than .4. If it did the label would be 0 and if it did not the label would be 1. The process of cleaning included dropping columns, rows that did not contain reviews, filtering out all non-english reviews, and fixing any misspellings. The feature engineering portion included processing the reviews into a vector, one hot encoding column with strings, scaling numeric features and then assembling them into one final feature vector column. I then ran a logistic regression model, giving me an accuracy of 65%. I then tuned the hyperparameters, but this did not increase the accuracy score.

Conclusions:

Overall, the model did okay, however there may be ways to improve the model. Through further refinement of the features, such as dropping unneeded columns and creating a feature from the reviews column. As well as further tuning hyperparameters.

A model like this can be used to filter out reviews which may not provide any benefit to a prospective buyer of a game. Such as a reviewer writing random letters and words. By detecting such issues, the model ensures that only meaningful and insightful reviews are presented, enhancing the decision-making process for potential buyers

Appendix A:

Api Command: kaggle datasets download -d kieranpoc/steam-reviews

Unzip File: unzip steam-reviews.zip

Create Bucket: gcloud storage buckets create gs://my-cis4130-rk --project=cis4130projectkabir \
--default-storage-class=STANDARD --location=us-central1 --uniform-bucket-level-access

Move File to Bucket: gsutil cp all_reviews.csv gs://my-cis4130-rk

Appendix B:

EDA Code:

```
file_path = 'gs://my-cis4130-rk/all_reviews.csv'

df_spark = spark.read.csv(file_path, header=True, inferSchema=True)
columns_to_select = ['game',
'author_num_reviews','author_num_games_owned','author_playtime_forever','language',
'review','timestamp_created','voted_up','votes_up','votes_funny','weighted_vote_score','comment
_count','steam_purchase','received_for_free']

df_selected = df_spark.select(columns_to_select)

df_selected.write.parquet("gs://my-cis4130-rk/my-data.parquet")
df_parquet = spark.read.parquet("gs://my-cis4130-rk/my-data.parquet")
```

I originally loaded the data into a regular spark dataframe from a csv, but then wrote it to a parquet to make processing time faster.

Get total records:
df_selected.count()

Get total null records & Visualize:

```
from pyspark.sql.functions import col, isnan, when, count
null_counts = df_selected.select([count(when(col(c).isNull() | isnan(c), c)).alias(c) for c in
df_selected.columns])

import matplotlib.pyplot as plt
import pandas as pd

null_counts_pandas = null_counts.toPandas()

null_counts_pandas = null_counts_pandas.T.reset_index()
null_counts_pandas.columns = ['Column', 'Null_Count']
plt.figure(figsize=(10, 6))
plt.bar(null_counts_pandas['Column'], null_counts_pandas['Null_Count'], color='skyblue')
#source:https://stackoverflow.com/questions/44627386/how-to-find-count-of-null-and-nan-value
s-for-each-column-in-a-pyspark-dataframe
plt.title('Null Value Counts by Column')
plt.xlabel('Column')
plt.ylabel('Number of Null Values')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

Review Language Count & Visualization:

```
grouped_df = df_selected.groupBy("language") \
    .agg(count("review").alias("review_count"))

top_10_languages = grouped_df.orderBy("review_count", ascending=False).limit(10)
import matplotlib.pyplot as plt
import pandas as pd

top_10_languages = top_10_languages.toPandas()
plt.figure(figsize=(10, 6))
plt.bar(top_10_languages['language'], top_10_languages['review_count'], color='skyblue')
```

```
plt.title('Reviews Counts by Language (10 millions)')
plt.xlabel('Review Count')
plt.ylabel('Number of Reviews')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

Oldest & Latest Review Time Stamp:

```
from pyspark.sql.functions import max, min

min_max = df_updated.agg(
    min("timestamp_created").alias("min_timestamp"),
    max("timestamp_created").alias("max_timestamp")
)
min_max.show()
```

Number of People who Received Game for Free:

```
from pyspark.sql.functions import count
grouped_sum_df =
df_updated.groupBy("game").agg(count("received_for_free").alias("total_sum"))
order_game = grouped_sum_df.orderBy("total_sum", ascending=False).limit(10)
order_game = order_game.toPandas()
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
plt.bar(order_game['game'], order_game['total_sum'], color='red')
plt.title('Number of People Who Received Game for Free (millions)')
plt.xlabel('Game')
plt.ylabel('Number of People Received Game for Free')
plt.xticks(rotation=45, ha='right')
plt.show()
```

Review Word Count:from pyspark.sql import functions as f

```
words_count = df_cleaned.withColumn('word', f.explode(f.split(f.col('review'), ' ')))\
    .groupBy('word')\
    .count()\
    .sort('count', ascending=False)\
```

```
    .show()
```

#Source:
https://stackoverflow.com/questions/48927271/count-number-of-words-in-a-spark-dataframe

Appendix C (Cleaning Code):

Changed data type for the timestamp column:

```
from pyspark.sql.functions import from_unixtime
df_updated = df_cleaned.withColumn("timestamp_created",
from_unixtime("timestamp_created"))
```

Fixed spelling of language column:

```
from pyspark.sql.functions import when

df_updated = df_updated.withColumn(
    "language",
    when(df_updated["language"] == "schinese", "chinese").otherwise(df_updated["language"])
)
df_updated = df_updated.withColumn(
    "language",
    when(df_updated["language"] == "koreana", "korean").otherwise(df_updated["language"])
)
```

Filter out the year for data from 2024 and earlier, as there are entry mistakes for the year:

```
from pyspark.sql.functions import year

df_with_year = df.withColumn("year", year(df["timestamp_created"]))
df_filtered = df_with_year.filter(year(df_with_year["timestamp_created"]) <= 2024)
```

Dropped all nans:

```
df_cleaned = df_filtered.na.drop()
```

Write to a new clean bucket:

```
output_path = "gs://my-cis4130-rk/clean-new.parquet"
df_cleaned.write.mode("overwrite").parquet(output_path)
```

Appendix D(Feature Engineering):

```
parquet_file_path = "gs://my-cis4130-rk/clean-new.parquet"
df = spark.read.parquet(parquet_file_path)
df = df.drop('timestamp_created', 'year', 'language','author_playtime_forever' )

columns = ["game", "author_num_reviews", "author_num_games_owned", "review",
"voted_up",
        "votes_up", "votes_funny", "weighted_vote_score", "comment_count",
        "steam_purchase", "received_for_free"]

for column in columns:
   df = df.filter(~col(column).isNull()).filter(~isnan(col(column)))




from pyspark.sql.functions import col
from pyspark.ml.feature import Binarizer

df = df.withColumn("weighted_vote_score", col("weighted_vote_score").cast("double"))

# creating the target variable

from pyspark.sql.functions import when

restored_df = restored_df.withColumn("weighted_vote_score",
col("weighted_vote_score").cast("double"))

# Create the binary column with a custom condition
restored_df = restored_df.withColumn(
   "target_variable",
   when(col("weighted_vote_score") <= .4, 1).otherwise(0)
)


game_titles = "gs://sample_data_games/steam_game_reviews_top_200_games.csv"
temp = spark.read.csv(game_titles,header=True)
filter_values = [row["game"] for row in temp.select("game").distinct().collect()]
```

```
df = df.filter(df["game"].isin(filter_values))

from pyspark.sql.functions import col

regex = r'^[^a-zA-Z]*$'

filtered_df = df.filter(
    (col("votes_up").rlike(regex)) & (col("votes_funny").rlike(regex)) &
(col("comment_count").rlike(regex)) & (col("steam_purchase").rlike(regex)) &
(col("received_for_free").rlike(regex))
)


df = df.fillna({"votes_up": 0, "votes_funny": 0, "author_num_games_owned": 0,
                "voted_up": 0, "steam_purchase": 0, "received_for_free": 0})

df = df.dropna(subset=["votes_up", "votes_funny", "author_num_games_owned", "voted_up",
"steam_purchase", "received_for_free"])

from pyspark.ml.feature import StringIndexer, OneHotEncoder, RegexTokenizer, HashingTF,
IDF, VectorAssembler, StandardScaler
from pyspark.ml.classification import LogisticRegression
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import BinaryClassificationEvaluator



# Step 1: Index and encode "game" column
indexer = StringIndexer(inputCol="game", outputCol="gameIndex")
encoder = OneHotEncoder(inputCols=["gameIndex"], outputCols=["gameVector"],
dropLast=False)

# Step 2: Tokenize "review" column, and compute TF-IDF
regexTokenizer = RegexTokenizer(inputCol="review", outputCol="words", pattern="\\w+",
gaps=False)
hashingTF = HashingTF(numFeatures=5000, inputCol="words", outputCol="word_features")
idf = IDF(inputCol="word_features", outputCol="tfidf_features", minDocFreq=2)

# Step 3: Assemble numeric features and scale them
assembler_scaling = VectorAssembler(
```

```
    inputCols=["votes_up", "votes_funny", "author_num_games_owned", "comment_count",
"author_num_reviews"],
    outputCol="scaled_assemble",
    handleInvalid="keep"
)
scaler = StandardScaler(inputCol="scaled_assemble", outputCol="scaled_features",
withMean=True, withStd=True)

# Step 4: Assemble all features into the final feature vector
assembler = VectorAssembler(
    inputCols=["scaled_features", "gameVector", "tfidf_features", "voted_up", "steam_purchase",
"received_for_free"],
    outputCol="final_features",
    handleInvalid="keep"
)


# Create the pipeline with all stages
feature_pipeline = Pipeline(stages=[indexer, encoder, regexTokenizer, hashingTF, idf,
assembler_scaling, scaler, assembler])

# Step 5: Fit the feature pipeline and transform the data
feature_pipeline_model = feature_pipeline.fit(df)
processed_df = feature_pipeline_model.transform(df)

checkpoint_dir = "gs://my-cis4130-rk/trusted/"
spark.sparkContext.setCheckpointDir(checkpoint_dir)

# Apply checkpointing to the DataFrame
checkpointed_df = processed_df.checkpoint()

# Save the checkpointed DataFrame as Parquet
checkpointed_df.write.parquet("gs://my-cis4130-rk/checkpoint/output-parquet/")
```

Initial Model Creation:

```
lr = LogisticRegression(
    featuresCol="final_features",
    labelCol="binary_vote_score")
```

```
predictions = lr_model.transform(test_data)

from pyspark.ml.evaluation import BinaryClassificationEvaluator

evaluator = BinaryClassificationEvaluator(labelCol="binary_vote_score",
rawPredictionCol="prediction")
accuracy = evaluator.evaluate(predictions)

print(f"Accuracy: {accuracy}")
```

Accuracy: 0.77685

Model on a smaller subset:

```
df_sampled = restored_df.sample(fraction=0.5)

# Split the data into training and test sets
train_data, test_data = df_sampled.randomSplit([0.8, 0.2])


lr = LogisticRegression(featuresCol="final_features", labelCol="binary_vote_score")

# Define the evaluator and hyperparameter grid
evaluator = BinaryClassificationEvaluator(labelCol="binary_vote_score")
paramGrid = (ParamGridBuilder()
        .addGrid(lr.regParam, [0.01, 0.1, 0.5])
        .addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0])
        .addGrid(lr.maxIter, [10, 50, 100])
        .build())

# Set up the CrossValidator
crossval = CrossValidator(estimator=lr,
                estimatorParamMaps=paramGrid,
                evaluator=evaluator,
                numFolds=3,
                parallelism=4)  # Parallelism for faster cross-validation

# Fit the model using CrossValidator
```

```
cv_model = crossval.fit(train_data)

# Get the best model
best_lr_model = cv_model.bestModel

# Print the best hyperparameters
print(f"Best regParam: {best_lr_model._java_obj.getRegParam()}")
print(f"Best elasticNetParam: {best_lr_model._java_obj.getElasticNetParam()}")
print(f"Best maxIter: {best_lr_model._java_obj.getMaxIter()}")

# Evaluate the best model on the test data
test_predictions = best_lr_model.transform(test_data)
auc = evaluator.evaluate(test_predictions)
print(f"Test AUC: {auc}")
```

```
Best regParam: 0.01
Best elasticNetParam: 0.0
Best maxIter: 50
```

Using new hyperparameters:

```
best_lr_model = LogisticRegression(
    featuresCol="final_features",
    labelCol="binary_vote_score",
    regParam=0.01,
    elasticNetParam=0.0,
    maxIter=50
)

trained_model = best_lr_model.fit(train_data)

predictions = trained_model.transform(test_data)

evaluator = BinaryClassificationEvaluator(labelCol="binary_vote_score",
metricName="areaUnderROC")
full_auc = evaluator.evaluate(predictions)

print(f" AUC: {full_auc}")
```

```
predictions = trained_model.transform(test_data)

evaluator = BinaryClassificationEvaluator(labelCol="binary_vote_score",
rawPredictionCol="prediction")
accuracy = evaluator.evaluate(predictions)

print(f"Accuracy: {accuracy}")
```

Full Dataset AUC: 0.6513028779119016

```
# Get coefficients (weights) from the logistic regression model
coefficients = trained_model.coefficients.toArray()

# Get the list of feature names (after transformations in the pipeline)
feature_names = [
    "scaled_features", "gameVector", "voted_up", "steam_purchase", "received_for_free"
]

# Print the features and their corresponding importance (coefficients)
feature_importance = list(zip(feature_names, coefficients))
feature_importance = sorted(feature_importance, key=lambda x: abs(x[1]), reverse=True)

for feature, importance in feature_importance:
    print(f"Feature: {feature}, Coefficient: {importance}")
```

Feature: voted_up, Coefficient: 0.07889746676435956
Feature: received_for_free, Coefficient: 0.0248937153522092
Feature: gameVector, Coefficient: 0.005741199168862145
Feature: scaled_features, Coefficient: 0.0057307063757713215
Feature: steam_purchase, Coefficient: 0.0027737313914948476

```
gcs_path = "gs://my-cis4130-rk/model"
# Save the trained model to GCS
best_lr_model.save(gcs_path)
```

Appendix E(Data Visualization):

Relationship plot:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Convert to a Pandas DataFrame
df = restored_df.select('votes_up', 'weighted_vote_score').limit(100000).toPandas()

# Set the style for Seaborn plots
sns.set_style("white")

# Create the relationship plot
lp = sns.lmplot(x='votes_up', y='weighted_vote_score', data=df)

# Add title and labels
lp.set(title="Votes Up vs Weighted Vote Score", xlabel="Votes Up", ylabel="Weighted Vote
Score")

# Save the plot
plt.savefig("votes_vs_weighted_score.png", dpi=300, bbox_inches='tight')

# Show the plot
plt.show()
```

Correlation Matrix:

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

# Convert the numeric values to vector columns
vector_column = "temp_col"
# Make a list of all of the numeric columns
numeric_columns = ['binary_vote_score','weighted_vote_score','author_num_reviews',
'author_num_games_owned', 'votes_up', 'votes_funny', 'comment_count']
# Use a vector assembler to combine all of the numeric columns together
assembler = VectorAssembler(inputCols=numeric_columns, outputCol=vector_column)
sdf_vector = assembler.transform(restored_df).select(vector_column)
```

```
# Create the correlation matrix, then get just the values and convert to a list
matrix = Correlation.corr(sdf_vector, vector_column).collect()[0][0]
correlation_matrix = matrix.toArray().tolist()
# Convert the correlation to a Pandas dataframe
correlation_matrix_df = pd.DataFrame(data=correlation_matrix, columns=numeric_columns,
index=numeric_columns)
sns.set_style("white")
# Create the plot using Seaborn
plt.figure(figsize=(16,5))
hm = sns.heatmap(correlation_matrix_df,
xticklabels=correlation_matrix_df.columns.values,
yticklabels=correlation_matrix_df.columns.values, cmap="Greens", annot=True)
figure = hm.get_figure()
figure.savefig("correlation_matrix.png", bbox_inches='tight')
```

Features Importance:

```
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming feature_importance is already calculated
# Convert the list of tuples into separate lists for plotting
features, importances = zip(*feature_importance)

# Plot the results
plt.figure(figsize=(10, 6))
sns.barplot(x=importances, y=features, palette="viridis")

# Add titles and labels
plt.title("Feature Importance from Logistic Regression Model", fontsize=16)
plt.xlabel("Coefficient Value (Importance)", fontsize=14)
plt.ylabel("Features", fontsize=14)

# Add grid lines for better readability
plt.grid(axis="x", linestyle="--", alpha=0.7)

plot_filename = "feature_importance.png"
plt.tight_layout()
plt.savefig(plot_filename, bbox_inches='tight', dpi=300)

# Show the plot
```

```
plt.tight_layout()
plt.show()
```

Confusion Matrix:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Extract the predicted labels (prediction column) and actual labels (binary_vote_score column)
y_pred = predictions_pandas['prediction']
y_true = predictions_pandas['binary_vote_score']

# Calculate the confusion matrix
cm = confusion_matrix(y_true, y_pred)

# Create a heatmap for the confusion matrix
plt.figure(figsize=(6, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Predicted Negative",
"Predicted Positive"],
            yticklabels=["Actual Negative", "Actual Positive"], cbar=False)

# Add labels and title
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')

# Show the plot
plt.tight_layout()
plt.show()
```

ROC Curves:

```
import matplotlib.pyplot as plt
plt.figure(figsize=(5,5))
plt.plot(lr_model.summary.roc.select('FPR').collect(),
lr_model.summary.roc.select('TPR').collect())
plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
plt.title("ROC Curve")
```