# CIS 4130 – CMWA [26518]
# Big Data Technologies Semester ML Project

**Name: Sylvia Corvalan Orellana**

## Proposal

For this project I've chosen a Kaggle dataset containing data from IMDb reviews, which is available through the following link:

https://www.kaggle.com/datasets/ebiswas/imdb-review-dataset

This is a dataset created from IMDb, a large database focused on film, television and similar content reviewed by people. The data collected in this dataset is publicly available and consists of 5,571,499 total record provided by 1,699,310 users. It has 9 columns that provide insight into each review.

## Columns

| | |
|---|---|
| **review_id** | It is generated by IMBb and unique to each review |
| **reviewer** | Public identity or username of the reviewer |
| **movie** | It represents the name of the show (can be movie, tv series, etc) |
| **rating** | Rating of movie out of 10, can be None for older reviews |
| **review_summary** | Plain summary of the review |
| **review_date** | Date of the posted review |
| **spoiler_tag** | If 1 = spoiler & 0 = not spoiler |
| **review_detail** | Details of the review |
| **helpful** | list[0] people find the review helpful out of list[1] |

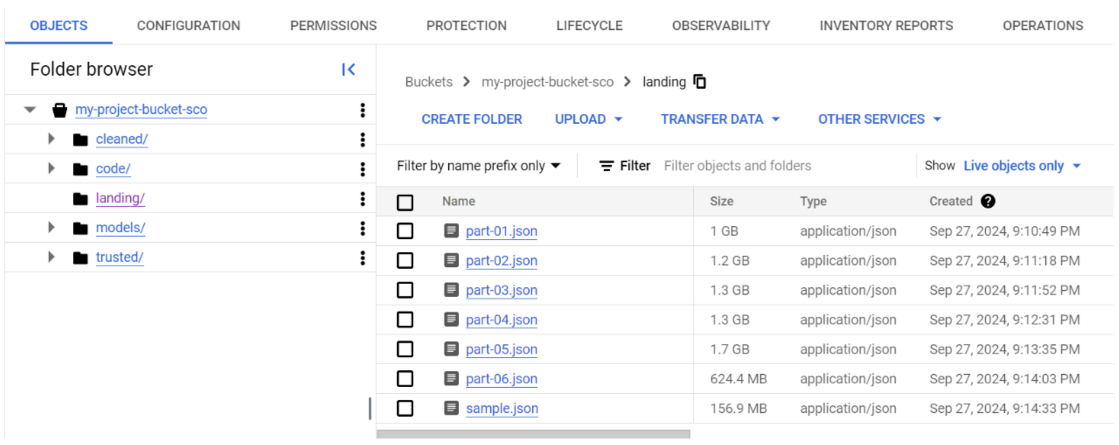
## What are we looking to predict?

We want to attempt to predict a movie or show's rating based on the reviewer, movie, review_date, spoiler tag, and the number of people who found the review helpful, as well as including the words used in the review itself using review_detail. This could be achieved by using a Logistic Regression model if we divide the rating into two categories: 'good', if it goes above a certain amount of stars, and 'bad', if it gets anything below that amount of stars.

## Data Acquisition

As this is a Kaggle dataset, this process was started by downloading the Kaggle API Token File and following the steps necessary to download the packages needed for this. Once that was done we followed by creating a bucket in Google Cloud Storage ('my-project-bucket-sco) using a Virtual Machine instance.

1. Creating the bucket:
2. Creating 5 folders (landing, cleaned, trusted, code, models):
3. Downloading Dataset from Kaggle
4. Unzipping to extract files
   - This results in 7 json files ('part-01.json', 'part-02.json', 'part-03.json', 'part-04.json', 'part-05.json', 'part-06.json', and 'sample.json')
5. Copying files to Google Cloud Storag



| Name | Size | Type | Created |
|---|---|---|---|
| part-01.json | 1 GB | application/json | Sep 27, 2024, 9:10:49 PM |
| part-02.json | 1.2 GB | application/json | Sep 27, 2024, 9:11:18 PM |
| part-03.json | 1.3 GB | application/json | Sep 27, 2024, 9:11:52 PM |
| part-04.json | 1.3 GB | application/json | Sep 27, 2024, 9:12:31 PM |
| part-05.json | 1.7 GB | application/json | Sep 27, 2024, 9:13:35 PM |
| part-06.json | 624.4 MB | application/json | Sep 27, 2024, 9:14:03 PM |
| sample.json | 156.9 MB | application/json | Sep 27, 2024, 9:14:33 PM |

## Exploratory Data Analysis

- **Number of records:** 5,671,499
- **Variables:**
  0 – review_id
  1 – reviewer
  2 – movie
  3 – rating
  4 – review_summary
  5 – review_date
  6 – spoiler_tag
  7 – review_detail
  8 – helpful
- **Rows with null values:** 674,941
- **Columns with null values:** Rating

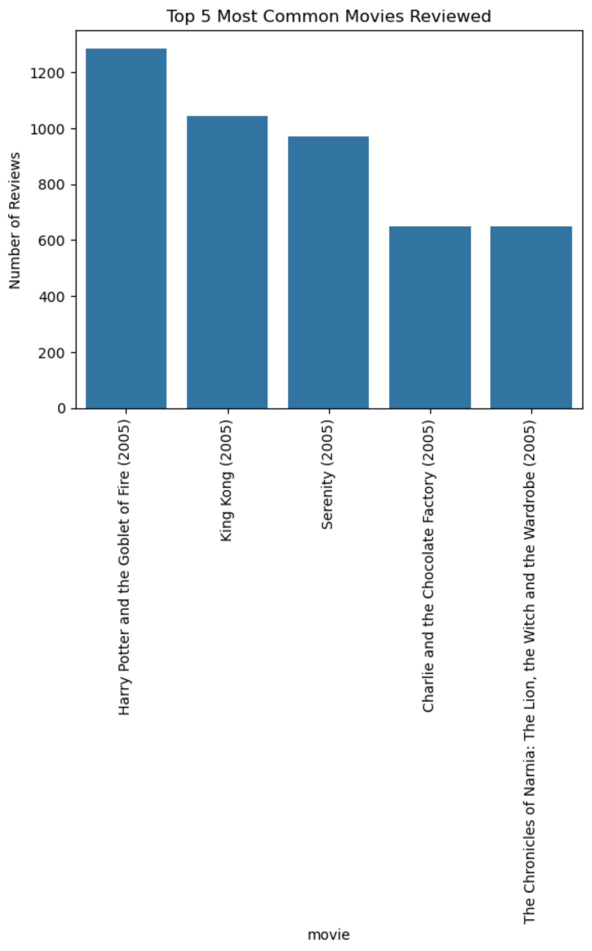
- **Min/Max/Mean/Std Dev:**

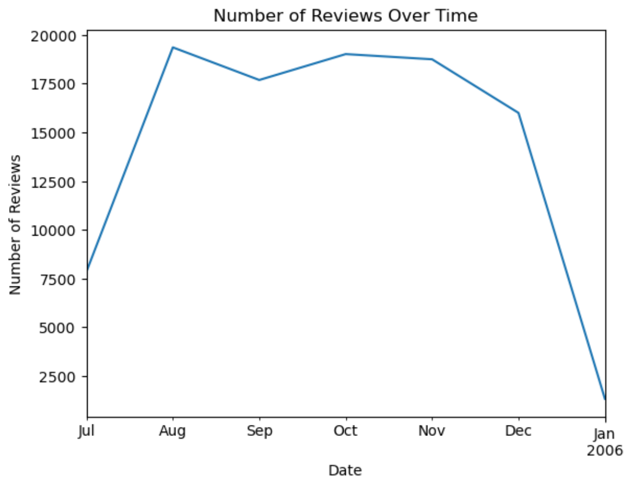
| | rating | review_date | spoiler_tag |
|---|---|---|---|
| Minimum | 1.00 | 1998-07-27 | 0.00 |
| Maximum | 10.00 | 2021-01-08 | 1.00 |
| Mean | 6.75 | - | 0.22 |
| Standard deviation | 2.99 | - | 0.42 |

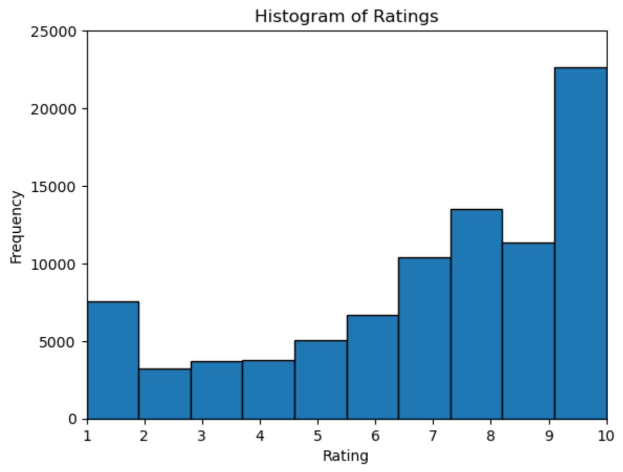
- **Word count statistics of columns review_summary and review_detail:**

| | review_summary word count | review_detail word count |
|---|---|---|
| Minimum | 1 | 28 |
| Maximum | 272 | 1108 |
| Mean | 5.88 | 244.95 |
| Standard deviation | 3.90 | 164.10 |

- **Plots**







We can see how the most reviewed movie is Harry Potter and the Goblet of Fire. We can also see that most reviews are made on August. Finally, ratings tend to be higher, most of them raging from 8 to 10 out of

## Data Cleaning

The steps we took to clean the data consisted on:

- Dropping null values on the rating column since rating is our target variable.
- Changing the date format so it would be easier to use
- Converting the rating into numeric variable instead of an object.
- Extracting the first element on the helpful column, as it was a list and it couldn't be used to see statistics on total people who found the reviews helpful.

## Expected Challenges

From this milestone, a potential challenge for future engineering would include finding the right variables to build the logistic regression model. It's necessary to find the correlations between the variables in order to choose between them.

## Feature Engineering

### Model

- Predict if rating is good or bad given movie, review_summary sentiment, and review_detail sentiment using Logistic Regression.

#### Features

| Column | Data Type | Variable Type | Indexer | Encoder |
|---|---|---|---|---|
| **movie** | string | Categorical | StringIndexer | OneHotEncoder |
| **summary_sentiment*** | float | Continuous | | |
| **detail_sentiment** | float | Continuous | | |

**summary_sentiment and detail_sentiment obtained from free text strings "review_summary" and "review_detail" by using VADER for sentiment analysis.*

**Label**

| Column | Data Type | Variable Type | |
|---|---|---|---|
| **rating** | float | Continuous | Label = 1.0 if rating > 6 Otherwise 0.0 |

## Summary of Program

1- Loading data from a parquet file from GCS bucket.
2- Feature engineering using VADER to get sentiment analysis on the review_summary and review_detail
3- Continuing feature engineering using a pipeline using StringIndexer and OneHoteEncoder to convert “movie” into numeric format. VectorAssembler used to combined features (movie_vector, detail_sentiment, summary_sentiment) inyo single vector column (features4) to use it in model.
4- Logistic Regression model trained on a sample of the transformed data set
5- Model is evaluated using accuracy, precision, recall, F1 score
6- Evaluating model using cross-validation and BinaryClassificationEvaluatior using “areUnderROC” metric.
7- Saving transformed data, model, and results to files in GCS.

## Summary of Outputs

**Confusion Matrix:**

| **Label** | **0.0** | **1.0** |
|---|---|---|
| **0.0** | **8672** | **4564** |
| **1.0** | **2935** | **19696** |

**True Negative:** 8671
**False Negative:** 2935
**False Positive:** 4564
**True Positive:** 19696

Accuracy ~ 0.7931

Precision ~ 0.8159

Recall ~ 0.8709

F1 score ~ 0.8425

These results are pretty good but could be improved, seems like the weakest portion is the accuracy.

**Average AUC-ROC score from cross validation over training data:**

~ 0.85397

**AUC-ROC score from model performance on test data:**

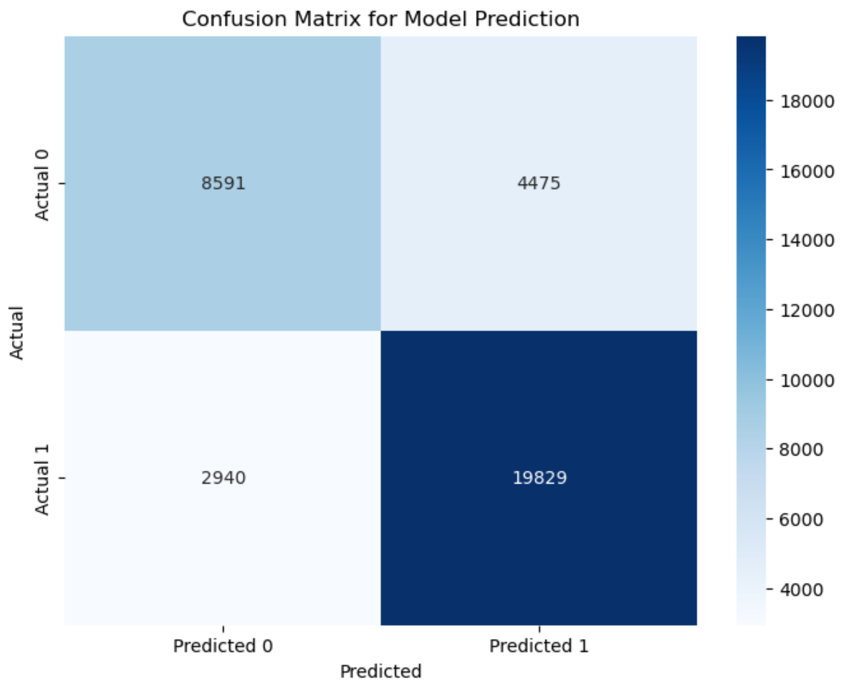
~ 0.8569

The AUC-ROC scores are high, so the model shows good performance based on this metric. The results seem consistent with the confusion matrix.

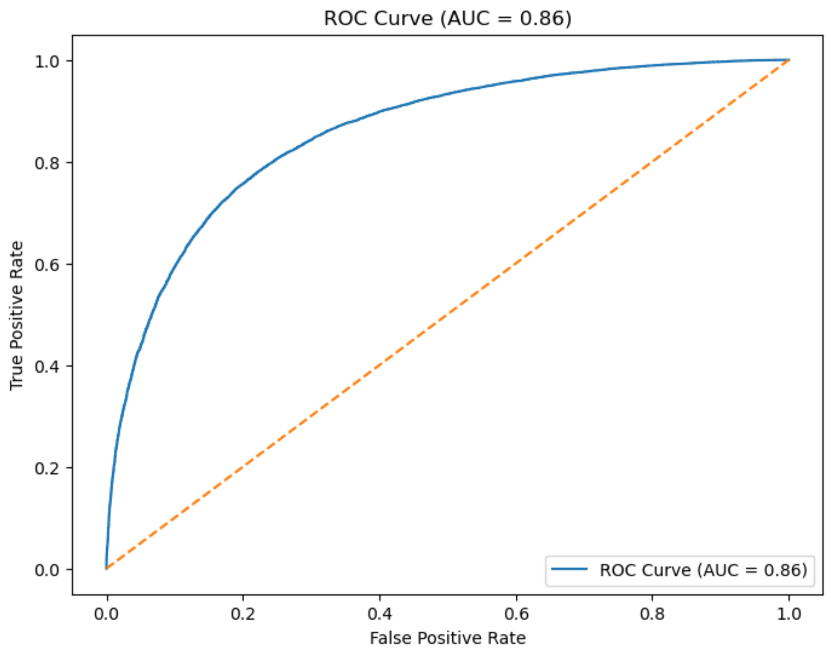
## Challenges Encountered

- The first challenge had to with the feature engineering of the review summary and detail, as review detail was long text and I was unsure with what technique was the best one, ended up doing a sentiment analysis to have a simplified numeric variable.
- Following up  the next challenge had to do with the “movie” variable as there too many  distinct movies because of the way that the title is entered into the review, this resulted in the movie vector being to large and the need to fix this issue by filtering the movies by the most frequent.

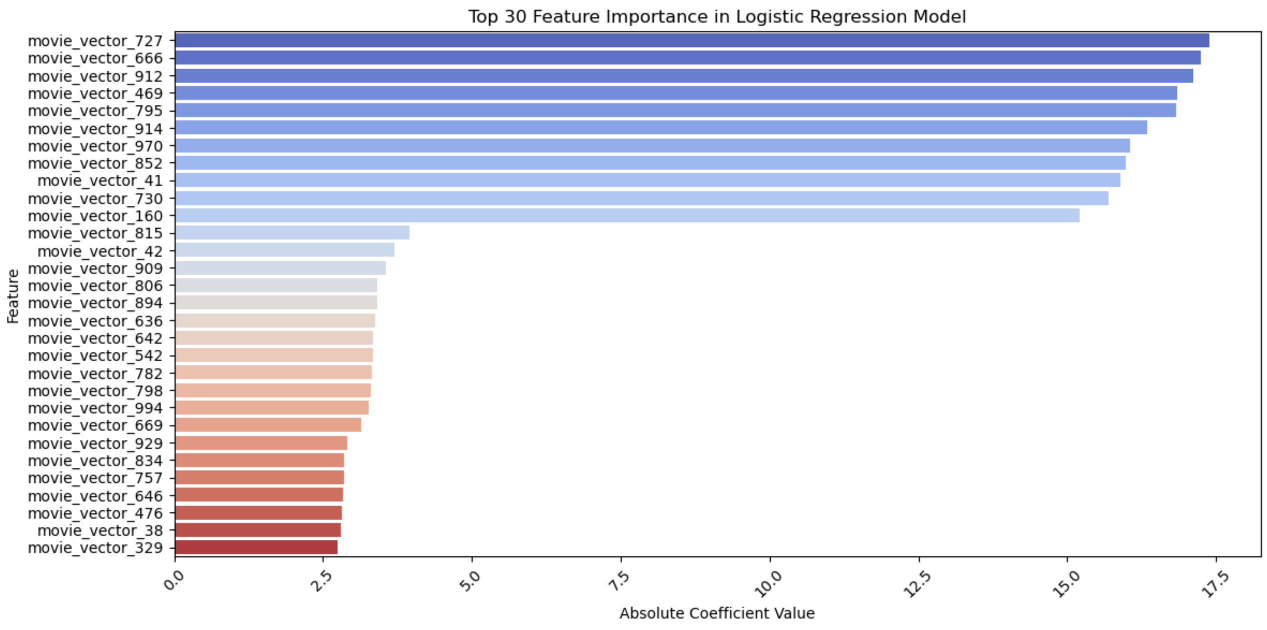
## Data Visualization



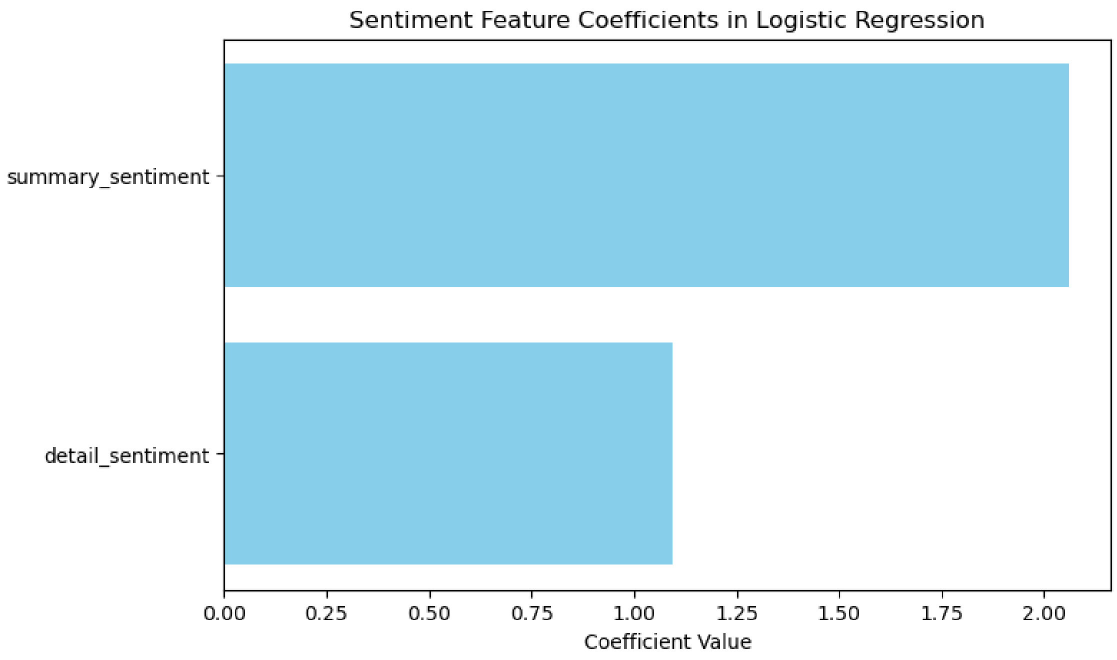
- By plotting the confusion matrix and using a heatmap we can visualize how the model is best at predicting positive labels.



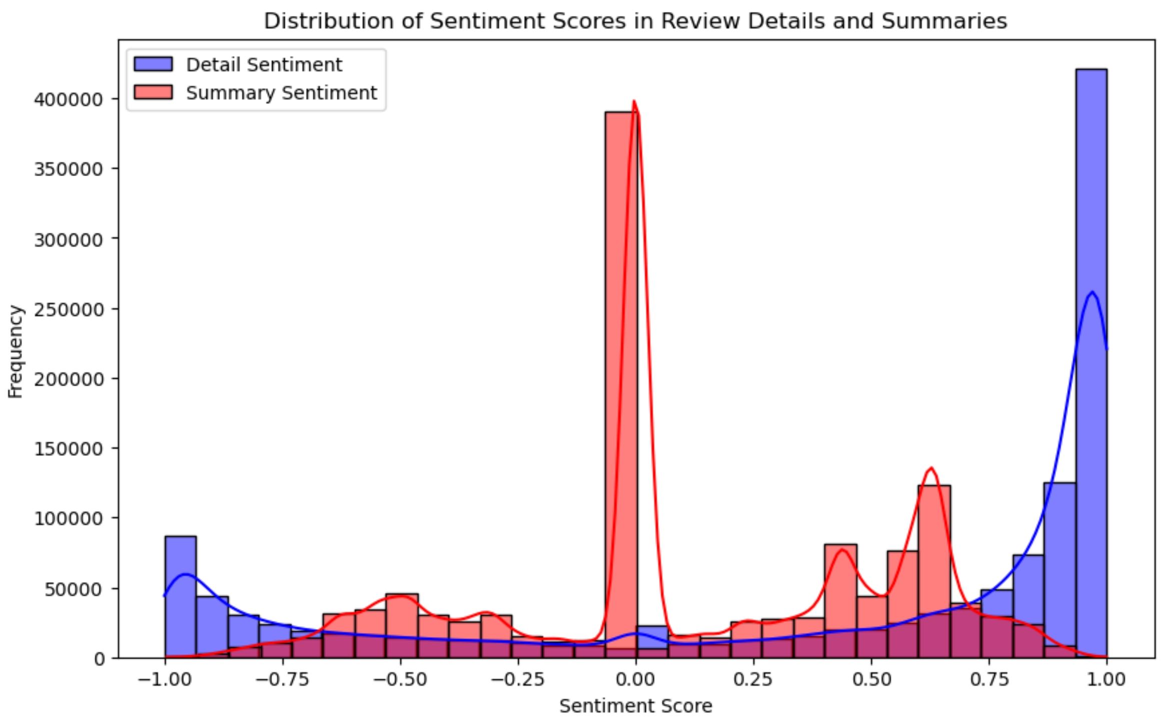
- This plot of the ROC Curve shows a good performance as it curves a way from the orange line representing a random classifier.



- This plot of the feature importance shows that the individual movies are the most important features, which is predictable as rating would heavily depend on the movie, but presents the issue of the model being overfitted to this particular set of movies, this should be improved for the model to be more useful.



- When extracting just the sentiments to see which one has more importance, we can see how the model is more sensitive to the summary_sentiment feature



- Finally this plot shows that summary_sentiment tends to be neutral while detail_sentiment tends to be more positive

## Summary of Data Processing Pipeline

1. **Data Acquisition:**
    - Dataset was sourced from Kaggle and downloaded using the Kaggle API. It was stored in a Google Cloud Storage (GCS) bucket and unzipped to extract seven JSON files, including 'sample.json'.
    - A Google Cloud bucket was created to store the dataset and five folders were set up for different stages of data processing: landing, cleaned, trusted, code, and models.
2. **Data Cleaning:**
    - Null Value Handling: Rows with missing values in the target column (rating) were dropped to ensure no missing data would interfere with the model.
    - Rating Conversion: The rating column was converted to a numeric format (from an object type) for better processing.
3. **Feature Engineering:**
    - Sentiment Analysis: VADER sentiment analysis was applied to both

    review_summary and review_detail columns to quantify their sentiment as numeric variables (positive, neutral, negative).
  - Categorical to Numeric: The movie column was converted into a numeric format using a StringIndexer and OneHotEncoder.
  - Feature Assembly: A VectorAssembler was used to combine multiple features (e.g., movie_vector, detail_sentiment, and summary_sentiment) into a single vector column, features4, which was then used as input for the model.
4. **Model Training:**
   - Logistic Regression: The logistic regression model was trained on a sample of the processed dataset to predict whether a review's rating is "good" or "bad".
   - Model Evaluation: The model's performance was evaluated using standard metrics, including accuracy, precision, recall, F1 score, and AUC-ROC score.
   - Cross-validation: Cross-validation was used to assess the model’s generalization ability, and a BinaryClassificationEvaluator was applied to calculate the AUC-ROC score.
5. **Results and Saving:**
   - The model achieved an accuracy of around 79%, precision of 81.6%, recall of 87.1%, and an F1 score of 84.3%. The AUC-ROC score was around 85.4% during cross-validation and 85.7% on the test data, indicating strong model performance.
   - The processed data, model, and results were saved to the GCS for further use.

## Conclusions of the Project

1. **Model Performance:**
   - The model performed well overall, with a relatively good AUC-ROC score, indicating good discriminatory ability between "good" and "bad" ratings. The accuracy, though reasonably high, could be improved, particularly in classifying positive ratings.
   - The confusion matrix and ROC curve visually supported the model's ability to differentiate between the classes, with the ROC curve curving away from the random classifier line.
2. **Feature Importance:**
   - The movie variable was found to be the most influential feature in predicting the rating, which is expected as movie ratings tend to be highly dependent on the movie itself. However, this also highlighted a potential overfitting issue since the model became very sensitive to specific movie titles.
   - Sentiment analysis of the review summary and detail showed that the model was particularly sensitive to the sentiment of the review summary, which was more neutral, while the review detail sentiment tended to be more positive.
3. **Challenges:**
   - Feature Engineering: One challenge was dealing with the long text in the review_detail column. A sentiment analysis approach was chosen to simplify the text into numerical variables, making it easier for the model to interpret.
   - Handling Movie Names: The large number of distinct movie titles in the dataset led to an issue with the movie feature, which resulted in a very large vector. Filtering the movie titles by frequency helped reduce this issue, but it may still affect the model’s generalizability.
   - Overfitting: The model's dependence on specific movie titles indicates a risk of overfitting, which means the model may not generalize well to new, unseen movies. This is an area that can be improved by selecting or encoding movies in a more robust way (e.g., using movie genres or grouping titles).
4. **Future Directions:**
   - The project could benefit from additional feature engineering, such as incorporating external data (e.g., movie genre, actor information) to diversify

the input features and reduce overfitting to specific movies.

- Testing other classification algorithms may also help improve accuracy and handle overfitting better.

## Appendix A: Data Acquisition Code

```
#Creating bucket
gcloud storage buckets create gs://my-project-bucket-sco-- project=cis4130 --
default-storage-class=STANDARD --location=us-central1
--uniform-bucket-level-access


#Creating 5 folders
gs://my-project-bucket-sco/landing/
gs://my-project-bucket-sco/cleaned/
gs://my-project-bucket-sco/trusted/
gs://my-project-bucket-sco/code/
gs://my-project-bucket-sco/models/


#Downloading dataset from Kaggle

mkdir .kaggle
mv kaggle.json .kaggle/
chmod 600 .kaggle/kaggle.json
kaggle datasets download -debiswas/imdb-review-dataset
pip3 install Kaggle

#Unzipping files
      sudo apt -y install zip
unzip imdb-review-dataset.zip
```

#Coping files to Google Cloud Storage

```
gcloud storage cp part-01.json gs://my-project-bucket-
sco/landing/part-01.json

gcloud storage cp part-02.json gs://my-project-bucket-
sco/landing/par-02.json

gcloud storage cp part-03.json gs://my-project-bucket-sco/landing/ part-
03.json

gcloud storage cp part-04.json gs://my-project-bucket-
sco/landing/part-04.json

gcloud storage cp part-05.json gs://my-project-bucket-
sco/landing/par-05.json

gcloud storage cp part-06.json gs://my-project-bucket-sco/landing/ part-
06.json

gcloud storage cp sample.json gs://my-project-bucket-sco/landing/sample.json
```

## Appendix B: Exploratory Data Analysis Code

```
# Import libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from google.cloud import storage
from io import StringIO
import numpy as np

# Set Pandas options to display floats with 2 decimal points (avoid scientific
notation)
pd.set_option('display.float_format', '{:.2f}'.format)
pd.set_option('display.width', 1000)

# Function to perform EDA
def perform_EDA(df, source_file_path):
    # Display basic info about the dataset
    print(f"\nPerforming EDA on file: {source_file_path}")
    print(df.info())
    print(df.head())

    # Convert 'review_date' column to datetime early in the process
    if 'review_date' in df.columns:
        df["review_date"] = pd.to_datetime(df["review_date"], errors='coerce')

    # Getting null values
    print("\nColumns with null values:",
df.columns[df.isnull().any()].tolist())
    print("Rows with null values:", df.isnull().any(axis=1).sum())

    # Getting summary statistics for numeric columns
    print("\nSummary statistics (numeric columns):")
    print(df.describe())

    # Word counts for review text columns
    df['summary_words'] = df['review_summary'].str.split().str.len()
    df['detail_words'] = df['review_detail'].str.split().str.len()
    print("\nSummary word count stats:")
    print(df['summary_words'].describe())
    print("Detail word count stats:")
    print(df['detail_words'].describe())

    # Plotting histograms for the rating column
    plt.hist(df['rating'], bins=10, edgecolor='black')
    plt.title('Histogram of Ratings')
    plt.xlabel('Rating')
    plt.ylabel('Frequency')
    plt.show()
```

```
      # Plotting reviews over time (if 'review_date' exists)
    if 'review_date' in df.columns:
        df.set_index('review_date', inplace=True)
        df.resample('M').size().plot()
        plt.title('Number of Reviews Over Time')
        plt.xlabel('Date')
        plt.ylabel('Number of Reviews')
        plt.show()

    # Show the top 5 most reviewed movies
    top_movies = df['movie'].value_counts().nlargest(5)
    sns.barplot(x=top_movies.index, y=top_movies.values)
    plt.title('Top 5 Most Common Movies Reviewed')
    plt.ylabel('Number of Reviews')
    plt.xticks(rotation=90)
    plt.show()

# Main function
def process_files_and_perform_EDA(source_bucket_name="my-project-bucket-sco",
folder_pattern="landing/"):
    # Initialize the Google Cloud Storage client
    client = storage.Client()

    blobs = client.list_blobs(source_bucket_name, prefix=folder_pattern)
    filtered_blobs = [blob for blob in blobs if blob.name.endswith('.json')]


    for blob in filtered_blobs:
        source_file_path =
f"gs://{source_bucket_name}/{folder_pattern}{blob.name}"


        df = pd.read_json(StringIO(blob.download_as_text()))

        perform_EDA(df, source_file_path)

# Initialize and run the EDA process
if __name__ == "__main__":
    # Specify your bucket name and folder pattern
    source_bucket_name = "my-project-bucket-sco"
    folder_pattern = "landing/"

    process_files_and_perform_EDA(source_bucket_name, folder_pattern)
```

# Appendix C: Data Cleaning Code

```
# Import libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from google.cloud import storage
from io import StringIO
import numpy as np

# Function to clean the data
def clean_data(df):
    # Drop rows with null values in 'rating' column
    df = df.dropna(axis=0, subset=['rating'])

    # Convert 'rating' to numeric
    df['rating'] = pd.to_numeric(df['rating'], errors='coerce')

    # Extract the helpful votes (assuming 'helpful' is in [0, 1] list)
    def extract_helpful_votes(helpful_list):
        return helpful_list[0] if isinstance(helpful_list, list) else 0

    df['helpful_count'] = df['helpful'].apply(extract_helpful_votes)

    # Return cleaned data
    return df

def process_and_clean(source_bucket_name="my-project-bucket-sco",
folder_pattern="landing/"):
    client = storage.Client()
    blobs = client.list_blobs(source_bucket_name, prefix=folder_pattern
    filtered_blobs = [blob for blob in blobs if blob.name.endswith('.json')]

    for blob in filtered_blobs:
        source_file_path =
f"gs://{source_bucket_name}/{folder_pattern}{blob.name}"

        df = pd.read_json(StringIO(blob.download_as_text()))

        # Clean the data
        df_cleaned = clean_data(df)

        # Generate the cleaned file name
        cleaned_file_name = blob.name.replace("landing/", "")

        # Set the output path in the "cleaned" folder
        output_file_path =
f"gs://{source_bucket_name}/cleaned/{cleaned_file_name}.parquet"

        # Save as Parquet file
        df_cleaned.to_parquet(output_file_path)
```

## Appendix D: Feature Engineering and Modeling Code

```
from pyspark import SparkContext
# pyspark.sql:
from pyspark.sql.functions import *
from pyspark.sql.types import FloatType

# pyspark.ml:
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler,
Tokenizer, HashingTF, IDF, StringIndexer, OneHotEncoder, Tokenizer, HashingTF,
IDF, VectorAssembler, MinMaxScaler, StandardScaler
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression,
LogisticRegressionModel
from pyspark.ml.evaluation import *
from pyspark.ml.tuning import *

import numpy as np
import nltk
nltk.download('vader_lexicon')
from nltk.sentiment import SentimentIntensityAnalyzer

# Read Parquet files
file_path = "gs://my-project-bucket-sco/cleaned/*.parquet"
df = spark.read.parquet(file_path)
# Show schema
df.printSchema()
# Show the first few rows of the dataframe
df.show(5)

# Initialize VADER sentiment analyzer
sia = SentimentIntensityAnalyzer()

# Define UDF that returns sentiment score from VADER
def get_sentiment_score_vader(text):
    if text:
        score = sia.polarity_scores(text)['compound']  # Compound score: ranges
from -1 to 1
        return score
    else:
        return 0.0

sentiment_udf = udf(get_sentiment_score_vader, FloatType())


# Apply the UDF
df = df.withColumn("detail_sentiment", sentiment_udf(col("review_detail")))

# Show results
df.show(5)
```

```
# Add summary sentiment column
df = df.withColumn("summary_sentiment", sentiment_udf(col("review_summary")))
df.show(5)

columns_to_drop = ["review_id", "reviewer", "review_summary", "review_detail",
"__index_level_0__"]
df = df.drop(*columns_to_drop)

df.show(5)

# Count movie frequency and sort
movie_frequency_df =
df.groupBy("movie").agg(count("movie").alias("frequency")).orderBy("frequency",
ascending=False)

# Show result
movie_frequency_df.show()


# We have what we want in movie_frequency_df
# Get the top 1000 movies
top_1000_movies_df = movie_frequency_df.limit(1000)

# Filter original DataFrame by doing an inner join with the top_1000_movies_df
on 'Movie' column
df = df.join(top_1000_movies_df, "movie")

# Show the result
df.show(5)


# Create a label.  =1 if rating > 6, =0 otherwise
df = df.withColumn("label", when(df.rating > 6, 1.0).otherwise(0.0) )
df.show(25)


# Set sample
sample_df = df.sample(fraction=0.1, seed=42)


# Define stages
indexer = StringIndexer(inputCols=["movie"],
                        outputCols=["movie_index"], handleInvalid='keep')

encoder = OneHotEncoder(inputCols=["movie_index"],
                        outputCols=["movie_vector"], dropLast=False)
assembler = VectorAssembler(inputCols=['movie_vector', 'detail_sentiment',
'summary_sentiment'],
                            outputCol="features4")
```

```
# Pipeline
ratings_pipe = Pipeline(stages=[indexer,encoder, assembler])
transformed_df = ratings_pipe.fit(sample_df).transform(sample_df)
transformed_df.select('movie','detail_sentiment','summary_sentiment', 'label',
'features4').show(5, truncate=False)

# Train/test split
trainingData, testData = transformed_df.randomSplit([0.7, 0.3], seed=42)


# Model fitting
lr = LogisticRegression(featuresCol="features4", labelCol="label")
model = lr.fit(trainingData)
print("Coefficients: ", model.coefficients)
print("Intercept: ", model.intercept)

test_results = model.transform(testData)
# Show test results
test_results.select('movie','detail_sentiment','summary_sentiment','rawPredicti
on','probability','prediction',
'label').show(truncate=False)


test_results.groupby('label').pivot('prediction').count().sort('label').show()


cm =
test_results.groupby('label').pivot('prediction').count().fillna(0).collect()
def calculate_recall_precision(cm):
    tn = cm[1]['0.0']  # True Negative: True label 0.0, predicted label 0.0
    fp = cm[1]['1.0']  # False Positive: True label 0.0, predicted label 1.0
    fn = cm[0]['0.0']  # False Negative: True label 1.0, predicted label 0.0
    tp = cm[0]['1.0']  # True Positive: True label 1.0, predicted label 1.0
    precision = tp / ( tp + fp )
    recall = tp / ( tp + fn )
    accuracy = ( tp + tn ) / ( tp + tn + fp + fn )
    f1_score = 2 * ( ( precision * recall ) / ( precision + recall ) )
    return accuracy, precision, recall, f1_score
print( calculate_recall_precision(cm) )

# Create a BinaryClassificationEvaluator to evaluate how well the model works
evaluator = BinaryClassificationEvaluator(metricName="areaUnderROC")

# Create the parameter grid
grid = ParamGridBuilder().build()


# Create the CrossValidator
cv = CrossValidator(estimator=lr, estimatorParamMaps=grid, evaluator=evaluator,
numFolds=3 )

# Use the CrossValidator to Fit the training data
```

```
cv = cv.fit(trainingData)
# Show the average performance over the three folds
cv.avgMetrics


evaluator.evaluate(cv.transform(testData))

#Define path
output_data_path = "gs://my-project-bucket-
sco/trusted/transformed_data.parquet"

# Savetransformed DataFrame
transformed_df.write.parquet(output_data_path, mode="overwrite")


# Define path
output_model_path = "gs://my-project-bucket-
sco/models/logistic_regression_model"

# Save trained model
model.save(output_model_path)

# Save test results to a CSV file
test_results.select('movie', 'detail_sentiment', 'summary_sentiment',
'rawPrediction', 'probability', 'prediction', 'label') \
    .write.csv("gs://my-project-bucket-sco/results/test_results.csv",
header=True, mode="overwrite")

# Save confusion matrix to a CSV file
confusion_matrix_df = spark.createDataFrame(cm, ["label", "prediction",
"count"])
confusion_matrix_df.write.csv("gs://my-project-bucket-
sco/results/confusion_matrix.csv", header=True, mode="overwrite")
accuracy, precision, recall, f1_score = calculate_recall_precision(cm)

# Format the evaluation metrics into a string
evaluation_metrics = f"Accuracy: {accuracy}\nPrecision: {precision}\nRecall:
{recall}\nF1 Score: {f1_score}"

# Define the GCS bucket and file path
bucket_name = "my-project-bucket-sco"
file_path = "results/evaluation_metrics.txt"

# Reference to the GCS bucket
bucket = storage_client.bucket(bucket_name)

# Reference to the file path within the bucket
blob = bucket.blob(file_path)

# Upload the string as a file
blob.upload_from_string(evaluation_metrics)
)
```

## Appendix E: Feature Engineering and Modeling Code

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from sklearn.metrics import confusion_matrix, roc_curve, roc_auc_score

# List of feature names
sentiment_features = ['detail_sentiment', 'summary_sentiment']
movie_features = [f"movie_vector_{i}" for i in range(1000)]  # Example, adjust
this based on actual number of unique movies
all_feature_names = movie_features + sentiment_features  # Combine both
sentiment and movie vector features


# Confusion Matrix Function
def plot_confusion_matrix(test_results):
    cm_data = test_results.select('label', 'prediction').toPandas()
    cm = confusion_matrix(cm_data['label'], cm_data['prediction'])

    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Predicted
0', 'Predicted 1'], yticklabels=['Actual 0', 'Actual 1'])
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Confusion Matrix for Model Prediction')
    plt.show()

# ROC Curve Function
def plot_roc_curve(test_results):
    # Convert rawPrediction vector to array for plotting
    from pyspark.ml.functions import vector_to_array
    test_results = test_results.withColumn("rawPredictionArray",
vector_to_array("rawPrediction"))

    # Extract the positive class probability (index 1)
    positive_class_prob =
test_results.select("rawPredictionArray").rdd.map(lambda row:
row[0][1]).collect()

    # Get the true labels
    true_labels = test_results.select("label").rdd.map(lambda row:
row[0]).collect()

    # Compute ROC curve
    fpr, tpr, thresholds = roc_curve(true_labels, positive_class_prob)

    # Calculate ROC AUC
    roc_auc = roc_auc_score(true_labels, positive_class_prob)

    # Plot ROC curve
    plt.figure(figsize=(8, 6))
    plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {roc_auc:.2f})")
    plt.plot([0, 1], [0, 1], linestyle="--")  # Random classifier diagonal
    plt.title(f"ROC Curve (AUC = {roc_auc:.2f})")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.legend(loc="lower right")
    plt.show()

# Feature Importance Function
def plot_feature_importance(model, all_feature_names, top_n=30):
    coefficients = model.coefficients
    coefficients_array = coefficients.toArray()

    # Create DataFrame of feature names and their coefficients
    coeff_df = pd.DataFrame({
        'Feature': all_feature_names,
        'Coefficient': coefficients_array
    })

    # Sort by absolute value of coefficient
    coeff_df['abs_coefficient'] = coeff_df['Coefficient'].abs()
    coeff_df = coeff_df.sort_values(by='abs_coefficient', ascending=False)

    # Select the top N features
    top_coeff_df = coeff_df.head(top_n)

    # Plot feature importance for top N features
    plt.figure(figsize=(12, 6))
    sns.barplot(x='abs_coefficient', y='Feature', data=top_coeff_df,
palette='coolwarm')
    plt.title(f'Top {top_n} Feature Importance in Logistic Regression Model')
    plt.xlabel('Absolute Coefficient Value')
    plt.ylabel('Feature')

    plt.xticks(rotation=45)
    plt.yticks(rotation=0)
    plt.tight_layout()
    plt.show()


# Sentiment Distribution Plot
def plot_sentiment_distribution(df):
    # Convert sentiment columns to pandas (this will move the data from Spark
to the local machine)
    sentiment_df = df.select("detail_sentiment",
"summary_sentiment").toPandas()

    # Plotting the distribution of sentiment scores
    plt.figure(figsize=(10, 6))
    sns.histplot(sentiment_df['detail_sentiment'], color='blue', kde=True,
label='Detail Sentiment', bins=30)
    sns.histplot(sentiment_df['summary_sentiment'], color='red', kde=True,
label='Summary Sentiment', bins=30)

    # Customize the plot
    plt.xlabel('Sentiment Score')
    plt.ylabel('Frequency')
    plt.title('Distribution of Sentiment Scores in Review Details and
Summaries')
    plt.legend()
    plt.show()

plot_confusion_matrix(test_results)
plot_roc_curve(test_results)
plot_feature_importance(model, all_feature_names)
plot_sentiment_distribution(df)
```