CIS 4130

Timothy Tran
TIMOTHY.TRAN@baruchmail.cuny.edu

Milestone 1

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

Summary:

The dataset contains user reviews from the IMDb website, primarily focusing on movie reviews. It has hundreds of thousands of reviews over time, providing insights into user sentiment, movie ratings, and helpfulness. Each row in the dataset holds detailed information such as the reviewer name,the movie, review text, user rating, and whether the review was marked as helpful.

Data Set Attributes:

- Review ID: Unique to each review (name)
- Movie: Represents name of movie/show
- Review Summary: Preview of the response provided by the user
- Review Date: The date in which the review was posted
- Review Rating: 1-10 integer of how the reviewer rated the movie
- Helpful Votes: The number of votes indicating whether the review was marked as helpful by other users.
- Review Detail: Actual response provided by reviewer
- Star Rating of Movie from Reviewer

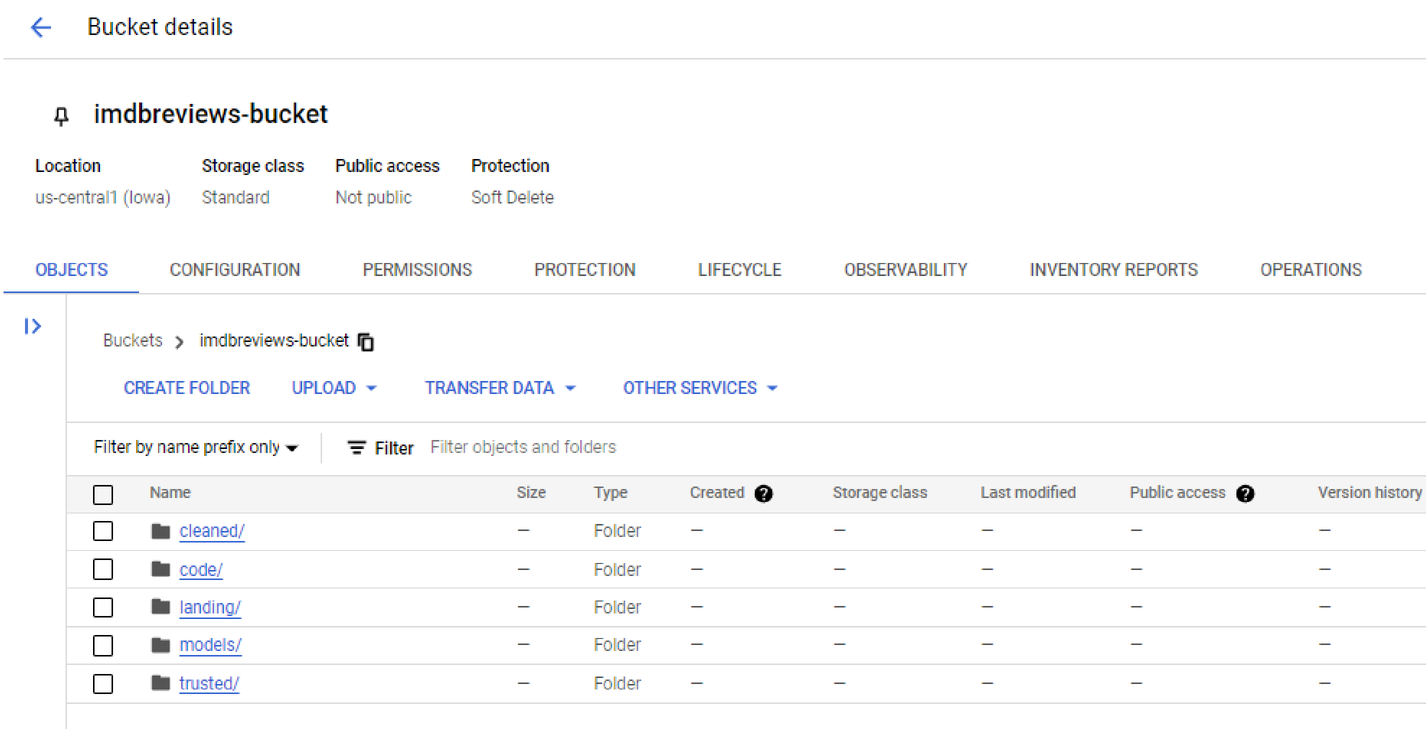
What I intend to predict:

I plan to predict the user rating on the movie. I will create a regression model that will utilize various features such as the review text, helpful votes, movie name, and release year to make predictions on the viewers rating of the movie.

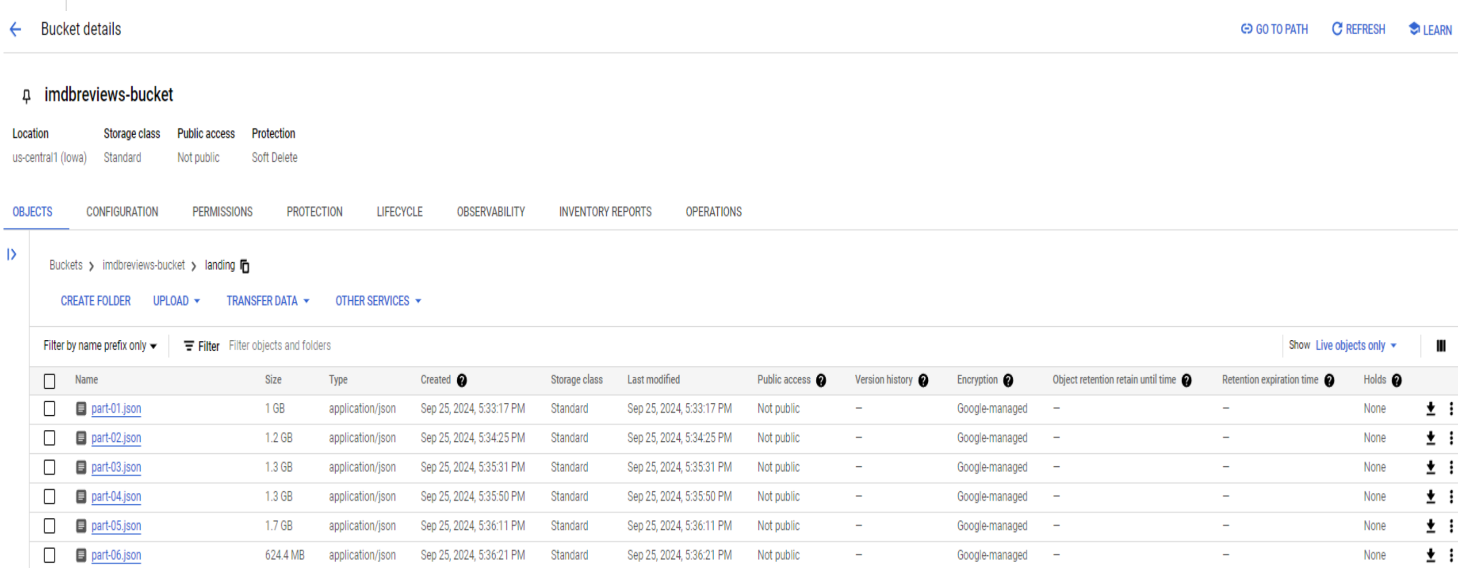
Milestone 2

Summary:

I created a bucket on Google Cloud called "imdbreviews-bucket" then within the bucket I created the folders cleaned, code, landing, models, and trusted. These folders would code and files that I would use to create my regression later on. I downloaded my kaggle dataset and imported the files to landing/.



Name	Size	Type	Created	Storage class	Last modified	Public access	Version history
cleaned/	–	Folder	–	–	–	–	–
code/	–	Folder	–	–	–	–	–
landing/	–	Folder	–	–	–	–	–
models/	–	Folder	–	–	–	–	–
trusted/	–	Folder	–	–	–	–	–



Name	Size	Type	Created	Storage class	Last modified	Public access	Version history	Encryption	Object retention retain until time	Retention expiration time	Holds
part-01.json	1 GB	application/json	Sep 25, 2024, 5:33:17 PM	Standard	Sep 25, 2024, 5:33:17 PM	Not public	–	Google-managed	–	–	None
part-02.json	1.2 GB	application/json	Sep 25, 2024, 5:34:25 PM	Standard	Sep 25, 2024, 5:34:25 PM	Not public	–	Google-managed	–	–	None
part-03.json	1.3 GB	application/json	Sep 25, 2024, 5:35:31 PM	Standard	Sep 25, 2024, 5:35:31 PM	Not public	–	Google-managed	–	–	None
part-04.json	1.3 GB	application/json	Sep 25, 2024, 5:35:50 PM	Standard	Sep 25, 2024, 5:35:50 PM	Not public	–	Google-managed	–	–	None
part-05.json	1.7 GB	application/json	Sep 25, 2024, 5:36:11 PM	Standard	Sep 25, 2024, 5:36:11 PM	Not public	–	Google-managed	–	–	None
part-06.json	624.4 MB	application/json	Sep 25, 2024, 5:36:21 PM	Standard	Sep 25, 2024, 5:36:21 PM	Not public	–	Google-managed	–	–	None

Milestone 3

Summary:
In Milestone 3, I conducted an Exploratory Data Analysis (EDA) using PySpark to identify the columns, variables, and missing values in the dataset. Additionally, I created a separate PySpark session to handle missing data and remove unnecessary columns. Finally, I saved the cleaned dataset as a Parquet file in the designated "cleaned" folder.

Subsection EDA:

HIGHLIGHTS and CONCLUSIONS:
In this section, the code connects to my Google Cloud bucket imdbreviews-bucket and lists JSON files in the landing/ folder. It was difficult to loop through the actual files but with the professors help and the use of python libraries that allowed me to import files from my google cloud storage. It would then perform an EDA on all the files and included: observations, listing variables, missing fields, and provided basic statistics for numeric columns using pandas (‘.describe()’).
Examples of the statistics were; In the first file, there are 1,010,293 observations with 51,520 missing ratings. The average rating is 6.70, and the average word count in the review details is 145.6, with a maximum of 3,339 words. The second file contains 1,012,212 observations, with 63,460 missing ratings. The average rating is 6.68, and review details have an average word count of 177.4, with a maximum of 2,720 words. The code would produce the same type of statistics for the 3rd, 4th, 5th, and 6th json files. At the end with the dataframe, I created a histogram that compared the average word count to the ratings given.

Example of Statistics Output:

```
Processing file: landing/part-01.json with size 1095550407 bytes
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1010293 entries, 0 to 1010292
Data columns (total 9 columns):
 #   Column          Non-Null Count    Dtype
---  ------          --------------    -----
 0   review_id       1010293 non-null  object
 1   reviewer        1010293 non-null  object
 2   movie           1010293 non-null  object
 3   rating          958773 non-null   float64
 4   review_summary  1010293 non-null  object
 5   review_date     1010293 non-null  object
 6   spoiler_tag     1010293 non-null  int64
 7   review_detail   1010293 non-null  object
 8   helpful         1010293 non-null  object
dtypes: float64(1), int64(1), object(7)
memory usage: 69.4+ MB
Number of observations: 1010293
```

```
Text data statistics:
review_id:
  - Number of documents: 1010293
  - Average word count: 1.0
  - Min word count: 1
  - Max word count: 1
reviewer:
  - Number of documents: 1010293
  - Average word count: 1.00339208526635345
  - Min word count: 1
  - Max word count: 6
movie:
  - Number of documents: 1010293
  - Average word count: 4.449136042712362
  - Min word count: 1
  - Max word count: 37
review_summary:
  - Number of documents: 1010293
  - Average word count: 5.121441007707665
  - Min word count: 0
```

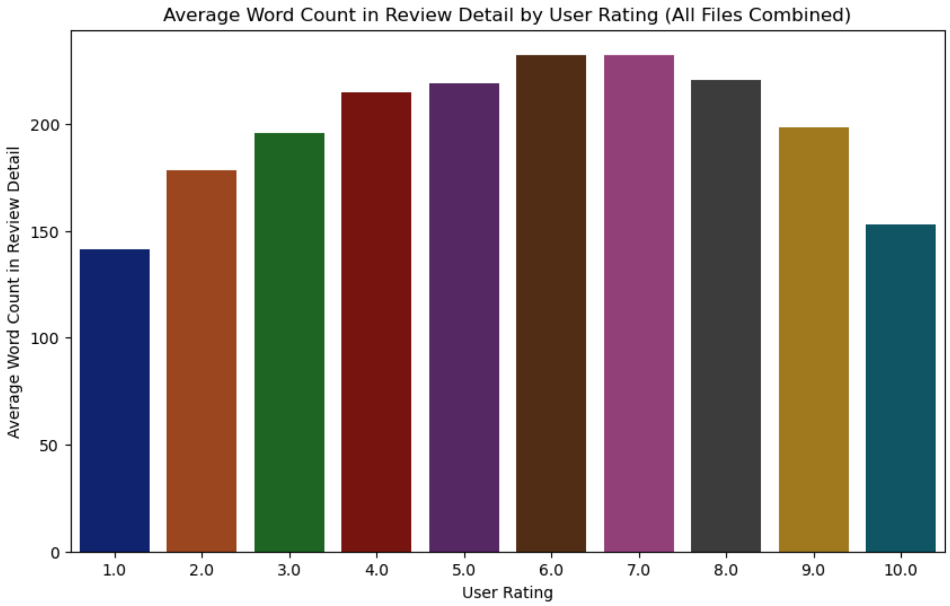
```
Statistics for numeric variables

Min values:
rating          1.0
spoiler_tag     0.0
Name: min, dtype: float64

Max values:
rating          10.0
spoiler_tag      1.0
Name: max, dtype: float64

Mean values:
rating          6.704052
spoiler_tag     0.187926
Name: mean, dtype: float64

Standard deviation:
rating          3.099434
```



Subsection DataCleaning:

For this code I first imported the necessary libraries, and then I created a variable with the bucket name that I will be reaching into and then a variable for the actually Google cloud storage. After I would create a function that would clean the data and fill in nulls and also remove and nulls that are not needed. After I would need to loop through all the files that is located in the GCS landing/ folder. Once that is grabbed it would then download the files as text and then read the .json file and create a dataframe as df. Then it would be sent to the cleaning function and run through. Lastly, the cleaned files.folder is sent to my cleaned/ folder in GCS as a parquet file. For challenges I think I will have in feature engineering, I believe it will be challenging to actually create the model and make it so there is small error and it will spit out accurate data. Really making it accurate and coding the model will be hard.

Milestone 4

TABLE:

Column Name	Data Type	Feature Engineering Treatment
movie	string	Categorical Data: Index and then Encode as a vector
review_summary	string	Text data: Sentiment Analysis
spoiler_tag	integer	Integer Data: Create as double
review_detail	string	Text data: Sentiment Analysis

Summary:

For the feature engineering portion, I had to find out the important columns I would be using for the randomforest regression and I removed the unnecessary columns like ‘reviewer’, ‘review_id’ , and ‘helpful’. Then with the important columns, I transformed the categorical data (movie) using StringIndexer and OneHotEncoder. For the review_detail and review_summary which are text_data I used a sentiment function from the module textblob. Then combined all the features into a single vector with VectorAssembler and put into a pipeline. For the model, I used Random Forest model. After splitting the data into training and testing sets, I evaluated the model's performance with calculations such as RMSE, MAE, and R2. I created a parquet for both features and models respectively in their folders.

One challenge I faced was dealing with the right columns to do feature engineering, especially around the review detail and summary. I had many problems dealing with running the feature engineering and the model because of the CPU usage and storage. I realized with the professor’s help it was due to the large amount of movies. So, the professor helped me and decided that 1000 movies should be fine. Overall, the pipeline helped me organize the process and columns. There is room to improve the feature engineering and tuning the movies to process more for better results.

```
Root Mean Squared Error (RMSE): 2.375414639919392
Mean Absolute Error (MAE): 1.8340007179369966
R-squared (R²): 0.446477903625073
```

Folder browser

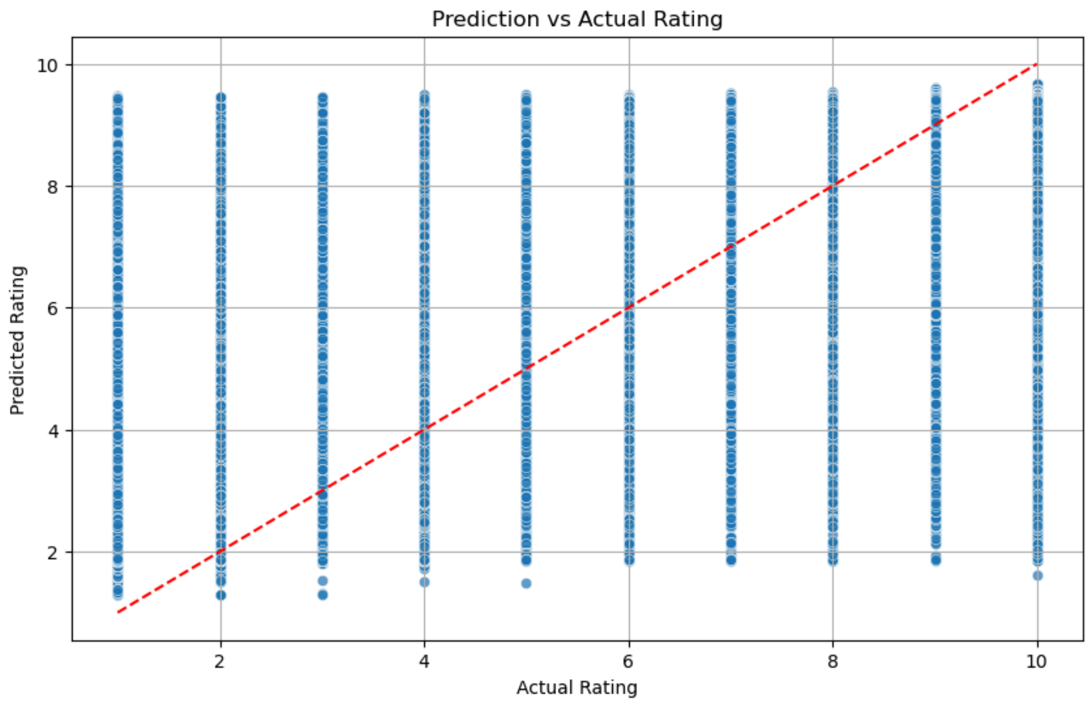
- imdbreviews-bucket
  - cleaned/
  - code/
  - features/
    - transformed_data_with_features.parquet/

```
Feature Importances:
Column<'movie'>: 0.006237457785416196
Column<'rating'>: 4.669557785406666e-05
Column<'review_date'>: 0.005325841866644272
Column<'spoiler_tag'>: 0.0019276701435910317
Column<'review_summary_sentiment'>: 0.008825961383685627
Column<'review_detail_sentiment'>: 3.704670377742539e-05
Column<'movie_index'>: 0.0022863716768154667
Column<'movie_vector'>: 0.0006293659283374953
Column<'features'>: 0.002423454561324252
```

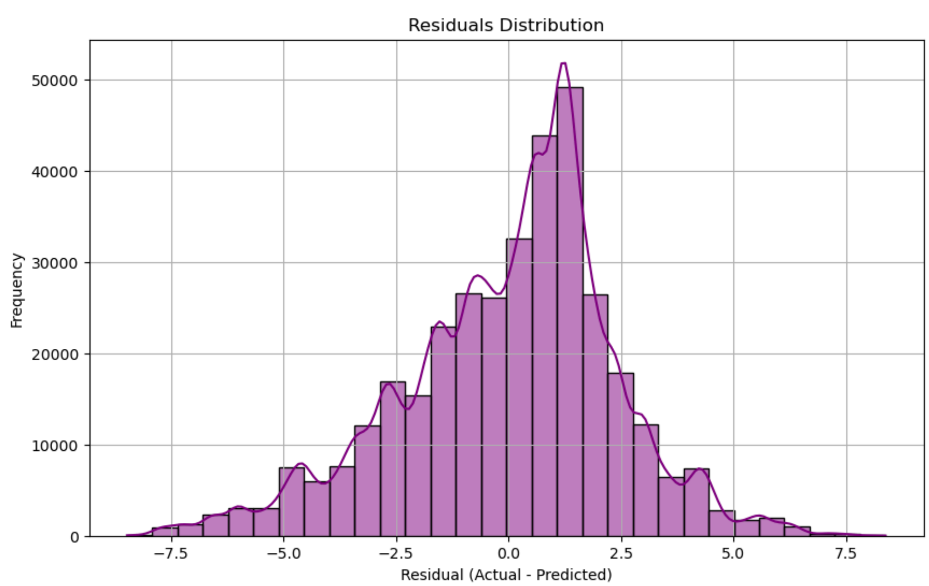
Milestone 5

Summary:

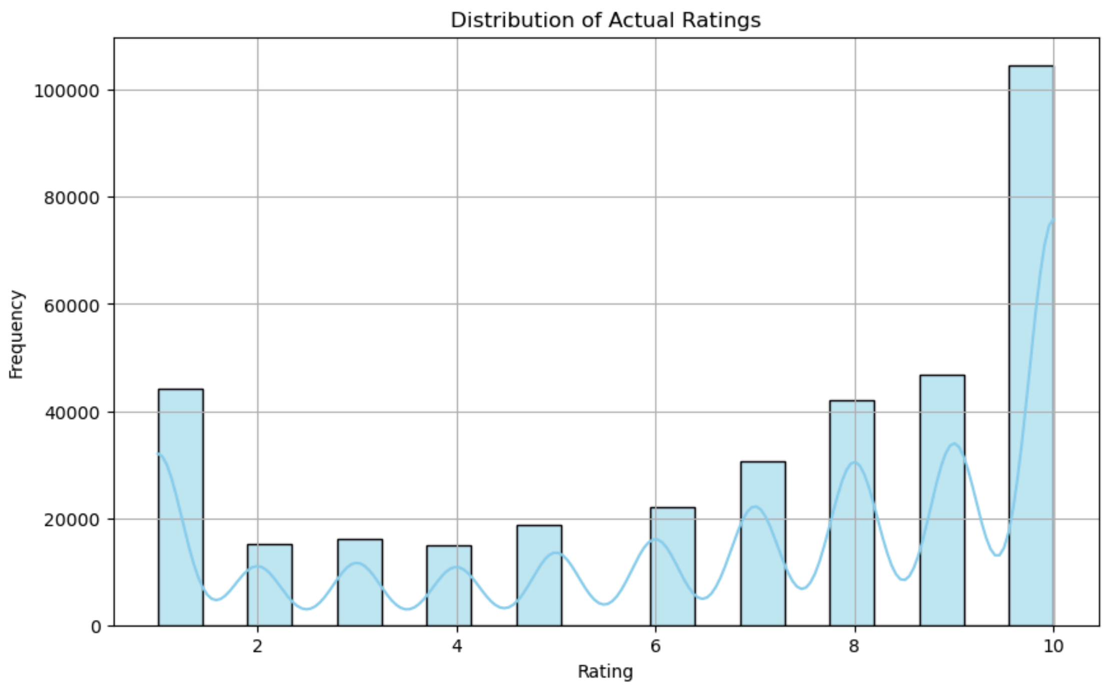
In this milestone I created 4 different visualisation graphs to showcase the different relationships and correlations between my features and the prediction results.



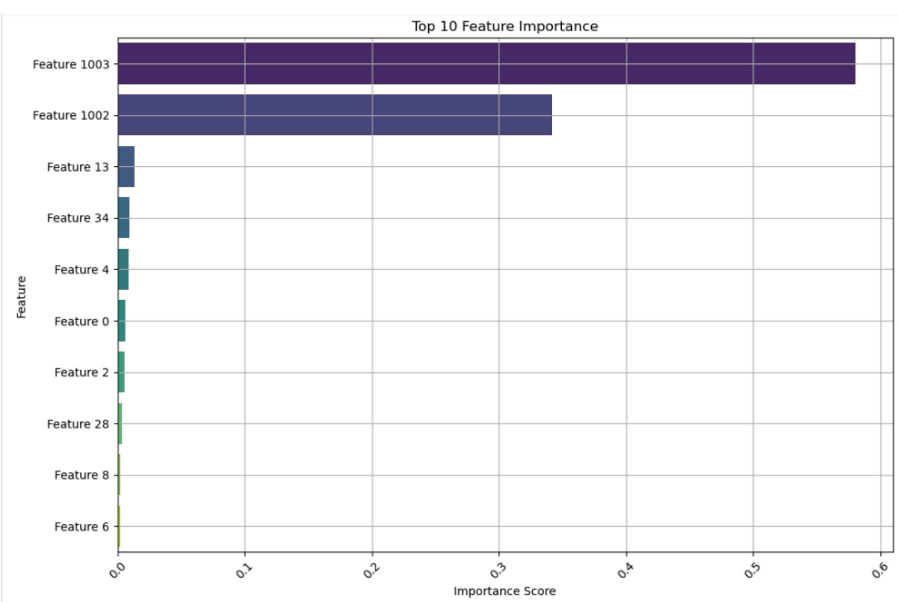
- I created a scatter plot to compare the predicted ratings to the actual ratings. The red dashed line represents perfect predictions, so I can see how close the model's predictions are to the real values. It helps me evaluate the overall accuracy and detect any patterns in the predictions.



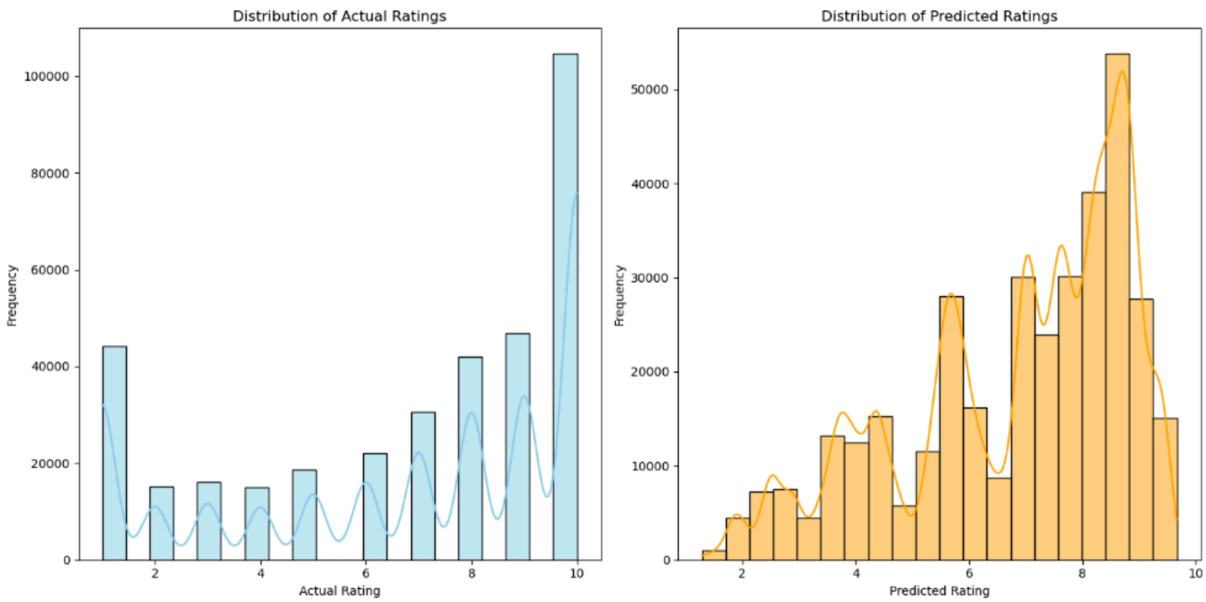
- I calculated the residuals (actual ratings minus predicted ratings) and plotted them as a histogram with a density curve. This shows how the errors are distributed, helping me identify if the model is over-predicting or under-predicting and whether the errors are concentrated or spread out.



- I plotted a histogram of the actual ratings to see how they are distributed in the dataset. This helps me understand whether the ratings are balanced, skewed, or concentrated in specific ranges, which can impact how well the model performs.



- I visualized the top 10 most important features from the Random Forest model using a bar chart. This highlights which features contribute the most to the predictions, helping me understand what drives the model's decisions and which factors are most significant in predicting movie ratings. Not sure why the names on the y-axis is feature #'s.



- I created side-by-side histograms to compare the distribution of actual and predicted ratings. On the left, I plotted the distribution of actual ratings using a sky-blue color, and on the right, I plotted the predicted ratings in orange. Both histograms has a curve to show the overall shape of the distributions. This comparison helps me see how closely the model's predictions align with the actual ratings and whether the distributions match or differ significantly.

Milestone 6

Summary:

In Milestone 6, I created a summary of the entire project and post the project details and the code in my GitHub repository for people to view.
Github website:https://timmyt571.github.io/CIS-4130-Semester-Project/

I first started by obtaining a dataset which I chose was IMDb reviews dataset from Kaggle (link) and creating a Google Cloud Storage (GCS) bucket called "imdb-reviews". Then created additional folders inside to organize my storage folders. Next, I set up clusters to create virtual instances and using PySpark to perform exploratory data analysis (EDA) on the dataset. This helped me identify key columns and address any null or missing values. I then created code for a cleaning version of the data to remove incomplete data and unnecessary columns. Moving on to feature engineering and modeling, I normalized the data, performed feature engineering, and chose to use random forest regression modeling to do training and testing. I allocated 70% of the data for training and 30% for testing. The processed data and trained models were stored in the models folder in my imdb-reviews bucket. Lastly, for data visualization, I used libraries like Matplotlib and Seaborn to create four visualizations that highlighted the dataset and its predictions.

Concluding my project and based on the visualizations and the data cleaning. We can predict that the distribution of actual ratings shows a clear peak at 10, indicating a user bias toward higher ratings, while this may not be a number, we can assume that ratings were mostly positive towards the movies. The predicted ratings capture this trend but with a smoother, more spread-out distribution. Although the model performs well in approximating the general pattern, it slightly underestimates the sharpness of the peak for the highest score. This could be due to the model averaging effects or the exclusion of other potentially influential factors such as parameters. To streamline the analysis, I focused on key attributes, removing excess data and working with a representative sample.

APPENDIX A

Getting started/Created Kaggle Token, uploaded it and searched kaggle dataset:

```
mkdir .kaggle
ls -la
mv kaggle.json .kaggle/
chmod 600 .kaggle/kaggle.json
```

Installing necessary things:

```
sudo apt -y install zip
sudo apt -y install python3-pip python3.11-venv
python3 -m venv pythondev
cd pythondev
source bin/activate
pip3 install kaggle
kaggle datasets list
```

Using CommandLine to download data set from kaggle:
kaggle datasets download -d ebiswas/imdb-review-dataset

The unzip the imdb-review-dataset.zip file using the command:
unzip imdb-review-dataset.zip

Create a bucket named 'my-bucket' in the us-central1 region within the project-id-12345 project ID.
gcloud storage buckets create gs://imdbreviews-bucket --project=thisisaproject-434114 --default-storage-class=STANDARD --location=us-central1 --uniform-bucket-level-access

Once the bucket is created, copy a file from the local file system to the new bucket:
gcloud storage cp part-01.json gs://imdbreviews-bucket/landing/part-01.json
gcloud storage cp part-02json gs://imdbreviews-bucket/landing/part-02.json
gcloud storage cp part-03.json gs://imdbreviews-bucket/landing/part-03.json
gcloud storage cp part-04.json gs://imdbreviews-bucket/landing/part-04.json
gcloud storage cp part-05.json gs://imdbreviews-bucket/landing/part-05.json
gcloud storage cp part-06.json gs://imdbreviews-bucket/landing/part-06.json

APPENDIX B

```
#Using Python to run to load the data set from GCS and produce descriptive statistics about the data.

from google.cloud import storage
from io import StringIO
import pandas as pd

bucket_name = "imdbreviews-bucket"
storage_client = storage.Client()
folder_pattern = "landing/"
blobs = storage_client.list_blobs(bucket_name, prefix=folder_pattern)
filtered_blobs = [blob for blob in blobs if blob.name.endswith('.json')]

#EDA on DataFrame
def perform_eda(df):
    if df.empty:
        print("No data")
        return
    #Number of observations
    num_observations = df.shape[0]
```

```
    print(f"Number of observations: {num_observations}")
    #List of variables
    print("List of variables (columns):")
    print(df.columns.tolist())

    missing_fields = df.isnull().sum()
    print("Number of missing fields in each column:")
    print(missing_fields[missing_fields > 0])

    #Statistics
    numeric_stats = df.describe()
    print("\nStatistics for numeric variables:")
    min_values = numeric_stats.loc['min']
    max_values = numeric_stats.loc['max']
    mean_values = numeric_stats.loc['mean']
    std_values = numeric_stats.loc['std']
    print("\nMin values:")
    print(min_values)
    print("\nMax values:")
    print(max_values)
    print("\nMean values:")
    print(mean_values)
    print("\nStandard deviation:")
    print(std_values)

    #Text statistics
    text_cols = df.select_dtypes(include=['object'])
    if not text_cols.empty:
        print()
        print("Text data statistics:")
        for col in text_cols.columns:
            if df[col].apply(lambda x: isinstance(x, str)).all():
                df['word_count'] = df[col].apply(lambda x: len(str(x).split()))
                print(f"{col}:")
                print(f" - Number of documents: {df[col].count()}")
                print(f" - Average word count: {df['word_count'].mean()}")
                print(f" - Min word count: {df['word_count'].min()}")
                print(f" - Max word count: {df['word_count'].max()}")
                df.drop('word_count', axis=1, inplace=True)

#Looping through the datafiles
for blob in filtered_blobs:
    print(f"Processing file: {blob.name} with size {blob.size} bytes")
    df = pd.read_json(StringIO(blob.download_as_text()))
```

```
    df.info()
    perform_eda(df)
    print()
    print()
```

#Creating the Histogram

```
from google.cloud import storage
from io import StringIO
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

source_bucket_name = "imdbreviews-bucket"
storage_client = storage.Client()
folder_pattern = "landing/"
blobs = storage_client.list_blobs(source_bucket_name, prefix=folder_pattern)
filtered_blobs = [blob for blob in blobs if blob.name.endswith('.json')]

#store data from all files in dataframe
all_data = pd.DataFrame()

#process each blob and append to main dataframe
for blob in filtered_blobs:
    print(f"Processing file: {blob.name} with size {blob.size} bytes")
    df = pd.read_json(StringIO(blob.download_as_text()))
    all_data = pd.concat([all_data, df], ignore_index=True)

#calculate word count for each 'review_detail'
all_data['word_count'] = all_data['review_detail'].apply(lambda x: len(str(x).split()) if pd.notnull(x) else
0)
# group by 'rating' and calculate the average word count
rating_word_count = all_data.groupby('rating')['word_count'].mean().reset_index()

#histogram of ratings with the average word count
plt.figure(figsize=(10, 6))
sns.barplot(data=rating_word_count, x='rating', y='word_count', palette='dark')
plt.xlabel('User Rating')
plt.ylabel('Average Word Count in Review Detail')
plt.title('Average Word Count in Review Detail by User Rating (All Files Combined)')
plt.show()
```

APPENDIX C
#Cleaning and moving files to /cleaned

```
from google.cloud import storage
from io import StringIO
import pandas as pd

#Source for the files
bucket_name = "imdbreviews-bucket"

#Create a client variable for GCS
storage_client = storage.Client()

#Get a list of the 'blobs' (objects or files) in the bucket
blobs = storage_client.list_blobs(bucket_name, prefix="landing")

#Data cleaning function
def clean_data(df):
    # Fill nulls or remove records with nulls
    df = df.fillna(value={"column_name": "default_value"})
    df = df.dropna()

    return df

#A for loop to go through all of the blobs and process each JSON file
for blob in blobs:
    if blob.name.endswith('.json'):
        print(f"Processing file: {blob.name}")

#CSV content into a DataFrame
        json_data = blob.download_as_text()
        df = pd.read_json(StringIO(json_data))

        #Print DataFrame info
        df.info()

        #Clean the data by calling the clean_data function
        df = clean_data(df)

        #Writing the cleaned DataFrame to the cleaned folder as a Parquet file
        cleaned_file_path = f"gs://{bucket_name}/cleaned/{blob.name.split('/')[-1].replace('.json',
'.parquet')}"
        df.to_parquet(cleaned_file_path, index=False)
```

```
print(f"Cleaned data written to: {cleaned_file_path}")
```

APPENDIX D

#Creating Features on columns (FEATURE ENGINEERING)

```
 #%pip install textblob
from google.cloud import storage
from pyspark.ml.feature import Tokenizer, RegexTokenizer, HashingTF, IDF, OneHotEncoder,
StringIndexer, VectorAssembler
from pyspark.sql.functions import col, monotonically_increasing_id
from pyspark.ml import Pipeline

df.printSchema()
# Drop columns we will not use at all
df = df.drop("review_id")
df = df.drop("reviewer")
df = df.drop("helpful")

from pyspark.sql.functions import count
# df.groupby("movie").count().show()
# Count frequency and sort
movie_frequency_df =
df.groupBy("movie").agg(count("movie").alias("frequency")).orderBy("frequency",
ascending=False)
# Show result
movie_frequency_df.show()
print(movie_frequency_df.count())

# 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")

# Drop the frequency column
df = df.drop("frequency")

# Show the result
df.show()
```

```
# Convert Spoler Tag to a double
df = df.withColumn("spoiler_tag", df.spoiler_tag.cast("double"))

indexer_movie = StringIndexer(inputCol="movie", outputCol="movie_index",
handleInvalid="keep")
encoder_movie = OneHotEncoder(inputCol="movie_index", outputCol="movie_vector",
handleInvalid="keep")

# sentiment analysis function
def sentiment_analysis(some_text):
    sentiment = TextBlob(some_text).sentiment.polarity
    return sentiment

#registering the UDF for sentiment analysis
sentiment_analysis_udf = udf(sentiment_analysis, DoubleType())

#apply the UDF to calculate sentiment for 'review_summary'
df = df.withColumn("review_summary_sentiment",
sentiment_analysis_udf(df["review_summary"]))
df = df.withColumn("review_detail_sentiment", sentiment_analysis_udf(df["review_detail"]))

#final feature vector
assembler = VectorAssembler(
    inputCols=[
#        "reviewer_vector",
        "spoiler_tag",
        "movie_vector",
        "review_summary_sentiment",
        "review_detail_sentiment"
    ],
    outputCol="features"
)

#pipeline with updated stages
pipeline = Pipeline(stages=[
    indexer_movie,
    encoder_movie,
    assembler
])

#fit and transform the pipeline on the data
df_transformed = pipeline.fit(df).transform(df)
```

```
# Drop unecessary columns
df_transformed = df_transformed.drop("helpful")
df_transformed = df_transformed.drop("review_summary")
df_transformed = df_transformed.drop("review_detail")

df_transformed.select("review_summary_sentiment", "review_detail_sentiment",
"movie_vector", "features").show(10, truncate=False)

df_transformed.cache()

# Save the transformed dataframe in a "features" folder
df_transformed.write.mode("overwrite").parquet(f"gs://imdbreviews-bucket/features/transformed
_data_with_features.parquet")
```

#Creating RandomForest Model

```
from google.cloud import storage
from pyspark.sql.functions import col
from pyspark.ml.regression import RandomForestRegressor
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

sdf =
spark.read.parquet("gs://imdbreviews-bucket/features/transformed_data_with_features.parquet"
)
sdf.show(10, truncate=False)


#RandomForestRegressor
rf = RandomForestRegressor(featuresCol="features", labelCol="rating")

#split data into training and test sets
train_data, test_data = sdf.randomSplit([0.7, 0.3], seed=42)

#set up cross-validation with hyperparameter tuning
paramGrid = ParamGridBuilder() \
    .addGrid(rf.numTrees, [10, 20, 30]) \
    .addGrid(rf.maxDepth, [5, 10, 15]) \
    .build()


#evaluate the model
```

```
evaluator_rmse = RegressionEvaluator(labelCol="rating", predictionCol="prediction",
metricName="rmse")
evaluator_mae = RegressionEvaluator(labelCol="rating", predictionCol="prediction",
metricName="mae")
evaluator_r2 = RegressionEvaluator(labelCol="rating", predictionCol="prediction",
metricName="r2")

cv = CrossValidator(estimator=rf,
                    estimatorParamMaps=paramGrid,
                    evaluator=evaluator_rmse,  # Evaluator for RMSE
                    numFolds=3)

#train the model
rf_model = cv.fit(train_data)

#make predictions on the test data
predictions = rf_model.transform(test_data)


rmse = evaluator_rmse.evaluate(predictions)
mae = evaluator_mae.evaluate(predictions)
r2 = evaluator_r2.evaluate(predictions)

print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"R-squared (R²): {r2}")

#sample predictions
predictions.select("movie","rating", "prediction").show(10, truncate=False)

best_model = rf_model.bestModel  # Best model after cross-validation

#Extract feature importances
feature_importances = best_model.featureImportances

#Print the feature importances
print("Feature Importances: ")
for feature, importance in zip(sdf, feature_importances):
    print(f"{feature}: {importance}")

#Save the trained model to a location
best_model.save("gs://imdbreviews-bucket/models/imdb_model")
```

```
#Save the predictions to models folder
predictions.select("movie","rating",
"prediction").write.parquet("gs://imdbreviews-bucket/models/rating_predictions")
```

APPENDIX E

```
from pyspark.ml.regression import RandomForestRegressionModel
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

model_path = "gs://imdbreviews-bucket/models/imdb_model"
rf_model = RandomForestRegressionModel.load(model_path)

# load the test predictions
rating_predictions_path = "gs://imdbreviews-bucket/models/rating_predictions/*"
predictions = spark.read.parquet(rating_predictions_path)

# convert predictions to Pandas
predictions_df = predictions.select("rating", "prediction").toPandas()

# Visualization 1: Scatter Plot (Prediction vs Actual)
plt.figure(figsize=(10, 6))
sns.scatterplot(x="rating", y="prediction", data=predictions_df, alpha=0.7)
plt.plot([predictions_df["rating"].min(), predictions_df["rating"].max()],
        [predictions_df["rating"].min(), predictions_df["rating"].max()],
        color='red', linestyle="--")
plt.title("Prediction vs Actual Rating")
plt.xlabel("Actual Rating")
plt.ylabel("Predicted Rating")
plt.grid(True)
plt.show()

# Visualization 2: Residual Distribution
predictions_df["residual"] = predictions_df["rating"] - predictions_df["prediction"]
plt.figure(figsize=(10, 6))
sns.histplot(predictions_df["residual"], kde=True, bins=30, color="purple")
plt.title("Residuals Distribution")
plt.xlabel("Residual (Actual - Predicted)")
plt.ylabel("Frequency")
plt.grid(True)
plt.show()

# Visualization 3: Actual Ratings Distribution
```

```
plt.figure(figsize=(10, 6))
sns.histplot(predictions_df["rating"], bins=20, kde=True, color="skyblue")
plt.title("Distribution of Actual Ratings")
plt.xlabel("Rating")
plt.ylabel("Frequency")
plt.grid(True)
plt.show()

# Visualization 4: Sort features by importance
top_features = importances.sort_values(by="importance", ascending=False).head(10)
# show top 10 features

# top features
plt.figure(figsize=(12, 8))
sns.barplot(x="importance", y="feature", data=top_features, palette="viridis")
plt.title("Top 10 Feature Importance")
plt.xlabel("Importance Score")
plt.ylabel("Feature")
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

# Visualization 5: Plot side-by-side histograms for actual and predicted ratings
plt.figure(figsize=(14, 7))

# Plot Actual Ratings Distribution
plt.subplot(1, 2, 1)
sns.histplot(predictions_df['rating'], kde=True, color='skyblue', bins=20, edgecolor='black')
plt.title('Distribution of Actual Ratings')
plt.xlabel('Actual Rating')
plt.ylabel('Frequency')

# Plot Predicted Ratings Distribution
plt.subplot(1, 2, 2)
sns.histplot(predictions_df['prediction'], kde=True, color='orange', bins=20, edgecolor='black')
plt.title('Distribution of Predicted Ratings')
plt.xlabel('Predicted Rating')
plt.ylabel('Frequency')

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