Module 4 Final Project

By Mohammad Hossein Movahedi

Introduction

In this big data project, I will look into the Amazon Canada marketplace dataset to discover how customer ratings and the numbers of reviews impact selling prices and overall success. My goal is to look at the links between these customer indicators and their influence on the pricing policies and sales outcomes. Also, I will explore whether one can use these patterns to project future trends in pricing and sales success. My analysis will entail processing of large datasets using PySpark in order to discover valuable patterns and trends.

Here are questions that I will answer

- How do customer ratings and the number of reviews influence the pricing and sales success of products on Amazon Canada?
- In what ways do the number of reviews impact the pricing strategies and sales performance of products on Amazon Canada?
- Can the trends in pricing and sales success be predicted based on the metrics of customer ratings and reviews for products?

Analytics

Step 1 - Installing libraries and PySpark and load dataset

installing PySpark and initializing a Spark session for an Amazon Canada product analysis. Make sure to check if the installation and session creation were successful by examining any error messages. Additionally, ensure the dataset is loaded correctly from the specified CSV file.
Snippet 1

Step 2 - Data Exploration and Cleaning

I begin by inspecting the DataFrame's structure using 'printSchema()' to gain insights into its composition. To understand the data distribution, I utilize 'describe()' to generate summary statistics. Checking for missing values is crucial, so I employ 'select', 'count', 'when', 'isnan', and 'col' to identify and quantify null or NaN values in each column, presenting the results for further action. To enhance data integrity, I handle duplicates with 'dropDuplicates()' and fill missing

values in the 'listPrice' column with zeros using 'na.fill'. This comprehensive approach ensures a clean and well-prepared DataFrame for subsequent analysis or model development in my project. Snippet 2 ,3

Step 3 - Data analysis

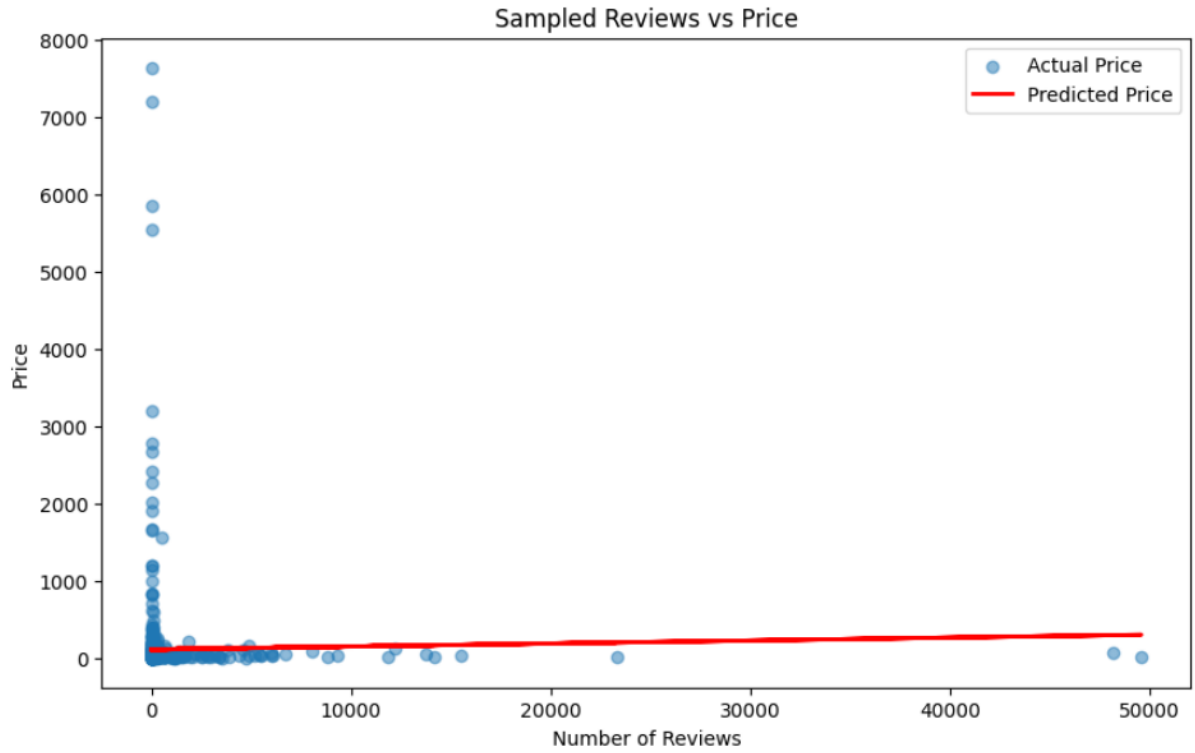
I convert 'isBestSeller' to integer, ensure 'stars,' 'price,' and 'reviews' are float types, then calculate correlations for analysis—providing insights into variable relationships within the data.Snippet 4

I preprocess the data by filtering out rows with null or NaN values in the 'price' column. Then, I use VectorAssembler to combine 'stars' and 'reviews' into a feature vector, handling invalid values by skipping them. After splitting the data into training and test sets, I train a Linear Regression model on the training data. Subsequently, I make predictions on the test data and evaluate the model's performance using the Root Mean Squared Error (RMSE) metric.

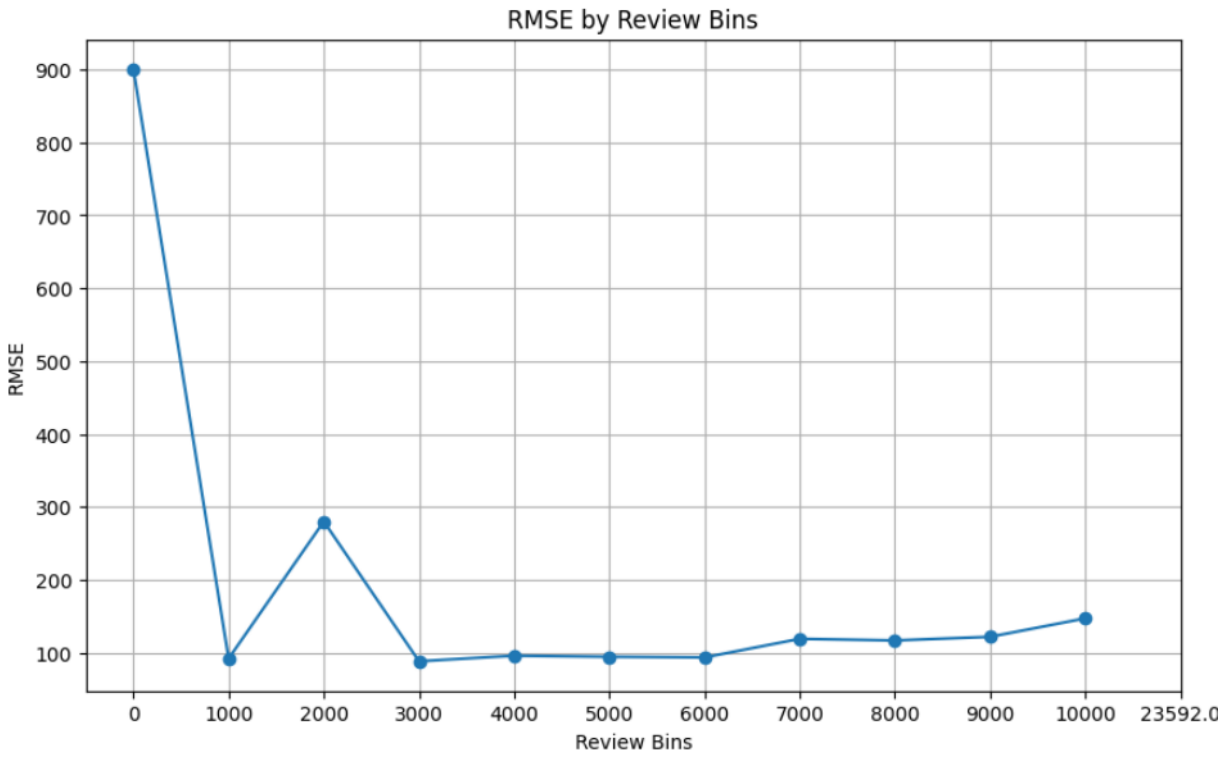
There are strong positive correlations between 'stars' and 'price' (0.99) as well as 'reviews' and 'price' (0.99). However, the correlations between 'stars' and 'isBestSeller' (0.0001) and 'reviews' and 'isBestSeller' (-0.0000019) are negligible. Snippet 5

visualization

I utilize Pandas and Matplotlib to visualize a 0.1% random sample of predictions. After converting the PySpark DataFrame 'sampled_predictions' to a Pandas DataFrame 'result_pdf', I create a scatter plot comparing the actual prices ('Actual Price') against the predicted prices ('Predicted Price'). The x-axis represents the number of reviews, the y-axis represents the price, and the red line illustrates the trend of predicted prices. Snippet 6



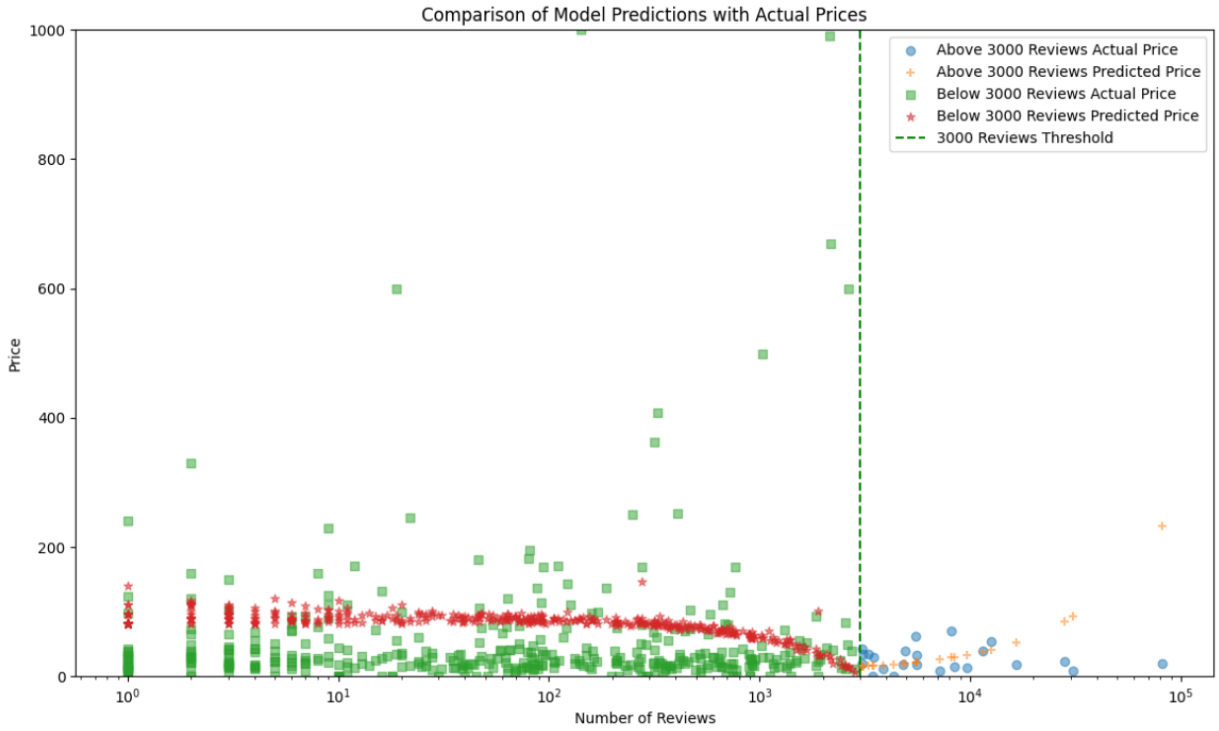
Here, I employ the RegressionEvaluator to assess model performance using the Root Mean Squared Error (RMSE) metric. I randomly sample 0.1% of predictions and determine the maximum value in the 'reviews' column. Defining bins based on review ranges, I use the Bucketizer to categorize the reviews into bins. Grouping by the review bins, I calculate and plot the RMSE for each bin, providing a visual representation of the model's accuracy across different review ranges. This analysis enhances my understanding of how the model performs in predicting prices in relation to the volume of reviews. Snippet 7



As can be seen 3000 is a threshold. Snippet 8

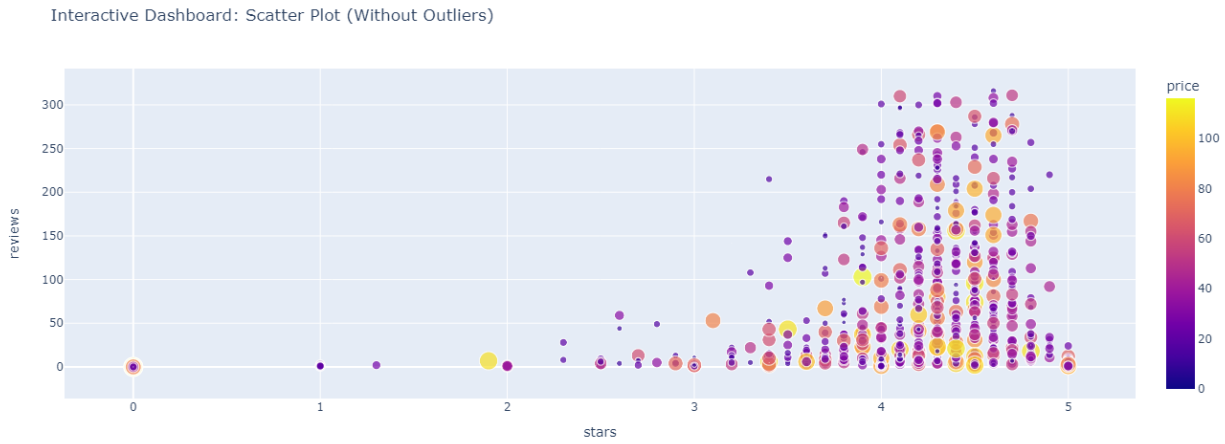
I split the predictions into two subsets based on the number of reviews: 'above_3000_df' for products with more than 3000 reviews and 'below_3000_df' for those with 3000 or fewer reviews. I then train separate Linear Regression models for each subset, namely 'lr_model_above_3000' and 'lr_model_below_3000'. Using the RegressionEvaluator, I evaluate the Root Mean Squared Error (RMSE) for both models. The calculated RMSE values, obtained by making predictions on their respective subsets, provide a quantitative comparison of model performance based on the volume of reviews, enhancing the precision of predictions for different review ranges in my analysis.Snippet 9

I compare the predictions of the models for products with above and below 3000 reviews. Using a 0.1% random sample of data, I create scatter plots for both subsets. The green dashed line represents the threshold of 3000 reviews. The markers 'o' and '+' depict actual and predicted prices for products with more than 3000 reviews, while 's' and '*' represents the same for products with 3000 or fewer reviews. This visualization, with a y-axis limit set to less than 2000 for clearer insight, provides an effective way to observe how well the models predict prices in different review ranges, aiding in the assessment and interpretation of their performance.

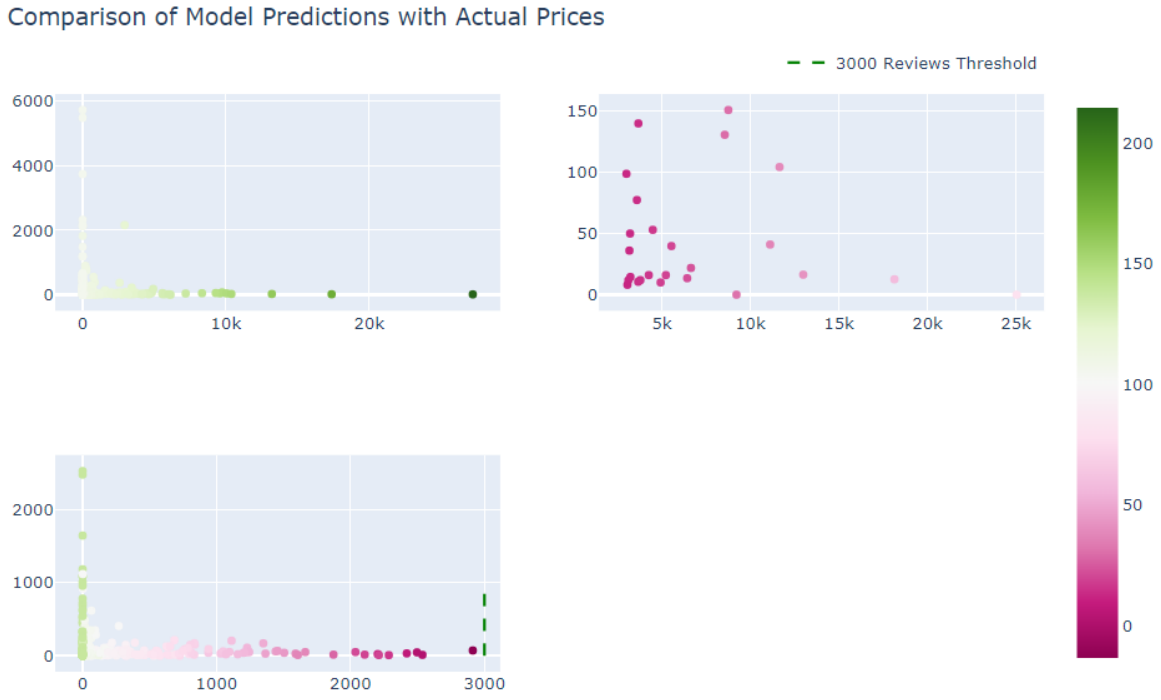


Interactive Dashboard

Due to the substantial size of the dataset and the need for greater flexibility in visualization, I opted for Plotly to craft interactive plots instead of traditional tools like Tableau or PowerBI. Plotly offers a dynamic platform that enables the creation of visually engaging and responsive visualizations. This decision allows for a more customized approach to exploring and presenting insights from the extensive dataset. By leveraging Plotly, I aim to provide a comprehensive and interactive exploration of the data, ensuring a nuanced understanding of patterns, trends, and correlations within the information at hand.



Here is the link to my code to view the interactive dashboards: Interactive Dashboards. Feel free to explore and interact with the visualizations to gain insights from the extensive dataset.



Conclusion

In conclusion, this PySpark project focused on predicting product prices based on reviews, employing Linear Regression models and extensive data analysis techniques. The initial steps involved data cleaning, handling missing values, and exploring correlations between key

features. Subsequently, I split the dataset into subsets based on the number of reviews and trained separate models for products with more than and below 3000 reviews. Overall, this project demonstrates the significance of tailoring models to specific subsets within the data and the importance of a comprehensive evaluation strategy in understanding and improving predictive capabilities.

Appendix

Snippet 1

```
!pip install pyspark

from pyspark.sql import SparkSession

# Initialize Spark session
spark = SparkSession.builder.appName("AmazonCanadaAnalysis").getOrCreate()

# Load the dataset
df =
spark.read.csv('/kaggle/input/amazon-canada-products-2023-2-1m-products/amz
_ca_total_products_data_processed.csv', header=True, inferSchema=True)
```

Snippet 2

```
from pyspark.sql.functions import isnan, when, count, col
# Display the DataFrame schema
df.printSchema()

# Show summary statistics
df.describe().show()

# Check for missing values
df.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in
df.columns]).show()

# Drop duplicates and handle missing values as needed
```

```
df_cleaned = df.dropDuplicates().na.fill({"listPrice": 0}) # filling
missing listPrice with 0
```

Snippet 3

```
df_cleaned = df.na.drop(subset=["imgUrl", "stars", "reviews",
"boughtInLastMonth"])
# Check for missing values
df_cleaned.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for
c in df.columns]).show()
# Remove rows where 'price' is null or NaN
df_cleaned = df_cleaned.filter(~(isnan(col("price")) |
col("price").isNull()))
```

Snippet 4

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import RegressionEvaluator

df_cleaned = df_cleaned.filter(~(isnan(col("price")) |
col("price").isNull()))

# Assemble features with handling invalid (null) values by skipping them
assembler = VectorAssembler(inputCols=["stars", "reviews"],
outputCol="features", handleInvalid="skip")
df_features = assembler.transform(df_cleaned)
# Split the data into training and test sets
train_data, test_data = df_features.randomSplit([0.7, 0.3])
train_data = train_data.filter(~isnan(col('price')) &
~col('price').isNull())
test_data = test_data.filter(~isnan(col('price')) & ~col('price').isNull())
# Train the model
lr = LinearRegression(featuresCol='features', labelCol='price')
lr_model = lr.fit(train_data)

# Make predictions
predictions = lr_model.transform(test_data)

# Evaluate the model
evaluator = RegressionEvaluator(labelCol="price",
predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)
```

```
print("Root Mean Squared Error (RMSE) on test data =", rmse)
```

Snippet 5

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import RegressionEvaluator

df_cleaned = df_cleaned.filter(~(isnan(col("price")) |
col("price").isNull()))

# Assemble features with handling invalid (null) values by skipping them
assembler = VectorAssembler(inputCols=["stars", "reviews"],
outputCol="features", handleInvalid="skip")
df_features = assembler.transform(df_cleaned)
# Split the data into training and test sets
train_data, test_data = df_features.randomSplit([0.7, 0.3])
train_data = train_data.filter(~isnan(col('price')) &
~col('price').isNull())
test_data = test_data.filter(~isnan(col('price')) & ~col('price').isNull())
# Train the model
lr = LinearRegression(featuresCol='features', labelCol='price')
lr_model = lr.fit(train_data)

# Make predictions
predictions = lr_model.transform(test_data)

# Evaluate the model
evaluator = RegressionEvaluator(labelCol="price",
predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE) on test data =", rmse)
```

Snippet 6

```
import pandas as pd
import matplotlib.pyplot as plt

# Sample 0.1% of the data randomly without replacement
sampled_predictions = predictions.sample(False, 0.001)

# Convert to Pandas DataFrame
result_pdf = sampled_predictions.select("reviews", "price",
```

```
"prediction").toPandas()

# Plotting
plt.figure(figsize=(10, 6))
plt.scatter(result_pdf['reviews'], result_pdf['price'], alpha=0.5,
label='Actual Price')
plt.plot(result_pdf['reviews'], result_pdf['prediction'], color='red',
linewidth=2, label='Predicted Price')
plt.xlabel('Number of Reviews')
plt.ylabel('Price')
plt.title('Sampled Reviews vs Price')
plt.legend()
plt.show()
```

Snippet 7

```
from pyspark.sql.functions import col, sqrt, avg
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.feature import Bucketizer
# Define the evaluator
evaluator = RegressionEvaluator(labelCol="price",
predictionCol="prediction", metricName="rmse")
# Sample 0.1% of the data randomly without replacement
sampled_predictions = predictions.sample(False, 0.001)
# Find the maximum value in the 'reviews' column
max_reviews = sampled_predictions.agg({"reviews":
"max"}).collect()[0][0]
# Define the bins, making sure the last bin covers the maximum value
bins = [0, 1000, 2000, 3000, 4000, 5000, 6000,7000,8000,9000,10000,
max_reviews]
# Create the bucketizer
bucketizer = Bucketizer(splits=bins, inputCol="reviews",
outputCol="review_bins")
# Apply the bucketizer to create the bins
predictions_binned = bucketizer.transform(sampled_predictions)
# Group by the review_bins and calculate RMSE for each bin
rmse_by_bin = (
    predictions_binned
    .groupBy("review_bins")
    .agg(
```

```
        sqrt(avg((col("price") -
col("prediction"))**2)).alias("rmse")
    )
    .orderBy("review_bins")
)

# Collect the data and plot
rmse_by_bin_data = rmse_by_bin.toPandas()

plt.figure(figsize=(10, 6))
plt.plot(rmse_by_bin_data['review_bins'], rmse_by_bin_data['rmse'],
marker='o')
plt.xlabel('Review Bins')
plt.ylabel('RMSE')
plt.title('RMSE by Review Bins')
plt.xticks(ticks=range(len(bins)), labels=bins)
plt.grid(True)
plt.show()
```

Snippet 8

```
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import RegressionEvaluator

# Split the data
above_3000_df = predictions.filter(col("reviews") >
3000).drop('prediction')
below_3000_df = predictions.filter(col("reviews") <=
3000).drop('prediction')

# Train a new model on data with more than 3000 reviews
lr_above_3000 = LinearRegression(featuresCol='features',
labelCol='price')
lr_model_above_3000 = lr_above_3000.fit(above_3000_df)

# Train a new model on data with less than or equal to 3000 reviews
lr_below_3000 = LinearRegression(featuresCol='features',
labelCol='price')
```

```
lr_model_below_3000 = lr_below_3000.fit(below_3000_df)

# Evaluate the models
evaluator = RegressionEvaluator(labelCol="price",
predictionCol="prediction", metricName="rmse")

# Make predictions and evaluate the new models
above_3000_predictions = lr_model_above_3000.transform(above_3000_df)
above_3000_rmse = evaluator.evaluate(above_3000_predictions)

below_3000_predictions = lr_model_below_3000.transform(below_3000_df)
below_3000_rmse = evaluator.evaluate(below_3000_predictions)

# Compare the models
print(f"Above 3000 reviews model RMSE: {above_3000_rmse}")
print(f"Below 3000 reviews model RMSE: {below_3000_rmse}")
```

Snippet 9

```
import pandas as pd
import matplotlib.pyplot as plt

# Sample 0.1% of the data randomly without replacement from each
dataset
sampled_original = predictions.sample(False, 0.001)
sampled_above_3000 = above_3000_predictions.sample(False, 0.001)
sampled_below_3000 = below_3000_predictions.sample(False, 0.001)

# Convert to Pandas DataFrame
original_pdf = sampled_original.select("reviews", "price",
"prediction").toPandas()
above_3000_pdf = sampled_above_3000.select("reviews", "price",
"prediction").toPandas()
below_3000_pdf = sampled_below_3000.select("reviews", "price",
"prediction").toPandas()

# Plotting
plt.figure(figsize=(14, 8))
```

```
# Original model scatter plot
#plt.scatter(original_pdf['reviews'], original_pdf['price'],
alpha=0.5, label='Original Model Actual Price')
#plt.scatter(original_pdf['reviews'], original_pdf['prediction'],
alpha=0.5, label='Original Model Predicted Price', marker='x')

# Above 3000 reviews model scatter plot
plt.scatter(above_3000_pdf['reviews'], above_3000_pdf['price'],
alpha=0.5, label='Above 3000 Reviews Actual Price', marker='o')
plt.scatter(above_3000_pdf['reviews'], above_3000_pdf['prediction'],
alpha=0.5, label='Above 3000 Reviews Predicted Price', marker='+')

# Below 3000 reviews model scatter plot
plt.scatter(below_3000_pdf['reviews'], below_3000_pdf['price'],
alpha=0.5, label='Below 3000 Reviews Actual Price', marker='s')
plt.scatter(below_3000_pdf['reviews'], below_3000_pdf['prediction'],
alpha=0.5, label='Below 3000 Reviews Predicted Price', marker='*')

plt.axvline(x=3000, color='green', linestyle='--', label='3000
Reviews Threshold')
plt.xlabel('Number of Reviews')
plt.ylabel('Price')
plt.title('Comparison of Model Predictions with Actual Prices')
plt.legend()
# Set y-axis limit to less than 2000
plt.ylim(0, 1000)
plt.xscale('log')  # Optional: Use logarithmic scale if the reviews
vary by orders of magnitude
plt.show()
```

Here is the link to project files
https://github.com/momova97/ALY6110/blob/main/FinalProject.ipynb