

CIS 4130 Term Project

Ching Hsuan, Lin

CIS 4130




[Chinghsuan.lin@baruchmail.cuny.edu](mailto:Chinghsuan.lin@baruchmail.cuny.edu)

## Milestone 1: project proposal

The dataset I choose is Amazon US Customer Review dataset. In this dataset it has several files depending on their product categories. In the each table of a category, it has columns like product\_id , customer\_id, review\_id, product\_category, star\_rating, review\_headline, and review\_body. For this project, I think I can used product review to do the sentiment analysis from natural language processing techniques to divide the reviews by positive and negative.

## Milestone 2: Data Acquisition

I transfer my dataset directly from Kaggle. I firstly set up my Command Line Interface in my EC2 instance. And I create a S3 bucket called mybucket4130. After that, I install the Kaggle command line interface. Then I copy my Kaggle API Token in the clipboard, and in the CLI I set up a Kaggle directory, create a new json file via nano editor, and paste my Kaggle API token then save and secure the file. One of the important steps is to edit two lines of kaggle\_api\_extended.py by using nano editor to accommodate writing the file to standard output. At the last step, I used the *Kaggle datasets list -s [KEYWORD]* to find my dataset then used *Kaggle datasets download -d [DATASET]* to download the dataset then append this line `aws s3 cp - s3://mybucket4130/amazonreview.zip` to directly transfer the dataset to my S3 bucket.

<input type="checkbox"/>	 amazonreview.zip	zip	September 24, 2022, 13:46:36 (UTC+08:00)	21.0 GB	Standard
<input type="checkbox"/>	 amazonreview/	Folder	-	-	-
<input type="checkbox"/>	 final_amz.csv	csv	October 20, 2022, 14:23:14 (UTC+08:00)	14.4 GB	Standard

## Milestone 3: Exploratory Data Analysis

The dataset has around 35 million rows of data. The dataset I used have 15 columns, includes [marketplace, customer\_id, review\_id, product\_id, product\_parent, product\_title, product\_category, star\_rating, helpful\_votes, total\_votes, vine, verified\_purchase, review\_headline, review\_body,review\_date] There are 5 numeric variables and 3 of them are id and the rest are helpful votes and total votes. I think the problem with cleaning and doing feature engineering would be some missing value, maybe some rows include some inconsistent data

type, and maybe run out of memory would be another main problem when I do the feature engineering.

Statistics of data: (some columns are ID so there is no need to get the statistics about them)

	A	B	C	D	E	F	G	H	I	J	K
1		star_rating	helpful_votes	total_votes	vine	verified_purchase		review_body	review_headline		review_date
2	missing values	41528	42299	43971	44976	45564		49224	46207		47793
3											
4	min	0	0	0			count	35,027,672	35,030,016	min	11/20/1998
5	max	5	41393	41889			unique	31,782,882	15,944,207	max	8/31/2015
6	average	4.1822013	1.51471	1.87281			top	Great	Five Stars		
7	standard deviation		19.43824	0.29896			freq	55,460	4,779,611		
8							number of words	1,728,360,240	145,907,071		
9											
10											

## Milestone 4: Coding and Modeling

For this Amazon customer review dataset, I'm going to do sentiment analysis from Amazon customer that I want to classify the attitude through customers' words in the reviews to see whether they are positive or negative to the products. In the beginning I would like to check all null values drop them since they are only small amount of data. The feature would be the customer reviews (review\_body), which I would need to do feature engineering to turn our raw review body into clean and without emoji and encode the customer rating to binary variable (positive, negative) and named the column "rating\_converted".

```
def ascii_only(mystring):
```

```
    if mystring:
```

```
        return mystring.encode('ascii', 'ignore').decode('ascii')
```

```
    else:
```

```
        return None
```

```
df = df.withColumn("rating_converted", when(col("star_rating") > 3,
'positive').otherwise('negative'))
```

I also create the machine learning pipeline which included tokenization, which converted sentence to into individual words. StopWordsRemover, which remove the stopwords such as "the", "an", and "at" to remove lower level information and let model to focus on important words. HashingTF, it can convert words into vectors format. And with IDF can extract the feature from the results of HashingTF, which provide us word frequency and can recognize how important within the documents. StringIndexer will help us to encode the rate to categorical variable such as positive become 0 and negative become 1. The last stage was the created the logistic regression. The reason I using this model because this regression was properly to be used when the dependent variable (label) has binary values. Besides, the results I'm going to predict is about customers'

emotion from their review to know whether their attitude was positive or negative, and I believe logistic regression would be good to deal with classification problem. Then I put all those steps into a pipeline The following code is the ML pipeline:

```
tokenizer = Tokenizer(inputCol="clean_review_body", outputCol="clean_review_words")
remover = StopWordsRemover(inputCol="clean_review_words", outputCol="remove_stopword")
hashtf = HashingTF(numFeatures=2**16, inputCol="remove_stopword", outputCol='tf')
idf = IDF(inputCol='tf', outputCol="features", minDocFreq=5)
label_stringIdx = StringIndexer(inputCol = "rating_converted", outputCol = "label")
lr = LogisticRegression(maxIter=100)
pipeline = Pipeline(stages=[tokenizer,remover, hashtf, idf, label_stringIdx, lr])
```

Before I fit the pipeline to training set, I tried to create grid search to find the most fitted hyperparameter which lead to best model performance.

```
grid = ParamGridBuilder()
grid = grid.addGrid(lr.regParam, [0.0, 1.0])
grid = grid.addGrid(lr.elasticNetParam, [0, 1])
```

Here we would create 4 models to be tested. And we use crossvalidator with the hyperparameter grid with the evaluator AUC:

```
evaluator = BinaryClassificationEvaluator(metricName="areaUnderROC")
cv = CrossValidator(estimator=pipeline, estimatorParamMaps=grid,evaluator=evaluator,
numFolds=3, seed=789)
```

After those steps, I can fit the model to training set and transform to test set and to use some evaluation function such as accuracy, precision, recall, f1 score.

```
>>> def calculate_recall_precision(cm):
...     fn = cm[0][1] # True Negative
...     tn = cm[0][2] # False Positive
...     tp = cm[1][1] # False Negative
...     fp = cm[1][2] # True Positive
...     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) )
(0.878308923971938, 0.9584011023792133, 0.8933681775537082, 0.9247426830938655)
```

Besides, we also have area under the ROC curve and confusion matrix, to see model performance, which will be showed in the visualization part. After evaluated the model, I save this file into parquet file then I convert it into csv to check some of the result and the prediction. The following are some of the results:

```

42 40 After only 4 months of wearing these shoes, the bottom sole split in half. I would never buy this again.

```

Above is the result of the prediction of shoes, it said “After only 4 months of wearing these shoes, the bottom sole split in half I would never but this again”. And is successfully classify as negative (1).

```

30795 30795 Love these. But don't leave them out in the hot summer sun. Mine warped and completely fell apart :-(

```

This one is kind of interesting since although they said “Love these”, but the rest of the sentence was talked about some flaws of the products, then the model classify it as negative (1).

```

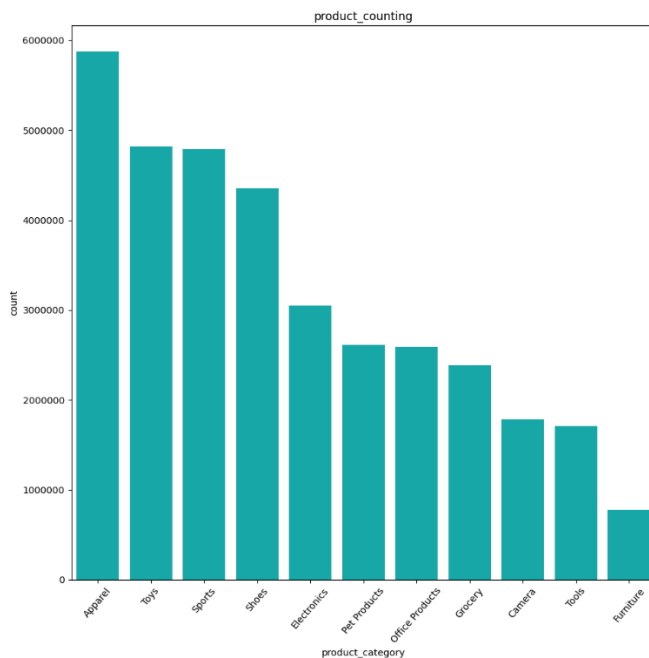
14830 14830 These sneakers are great for when I walk my Yorkies & very comfy. I'm glad I had them on when I had to take a Taxi & the driver had 80!! So I walked home after I was done at Travis Credit Union. I would have given 5 stars, but the laces are a bit on the short side.
30806 30806 Love these. Surprisingly comfortable and stylish.
30807 30807 Love these. They are fantastic. I have nothing negative to say about these at all. They are amazingly comfortable - so squishy underfoot - glorious! Other than being a hot sweaty mess after a run with the Hoka One One's I don't have various the dull aches I used to get

```

Above are some other example of the positive (0) results.

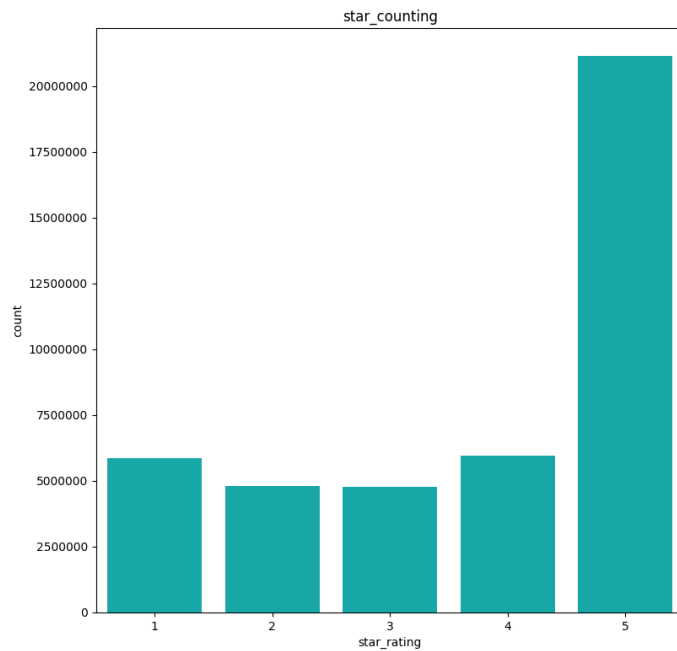
## Milestone 5: Visualization

### 1. Product counting graph (EDA)



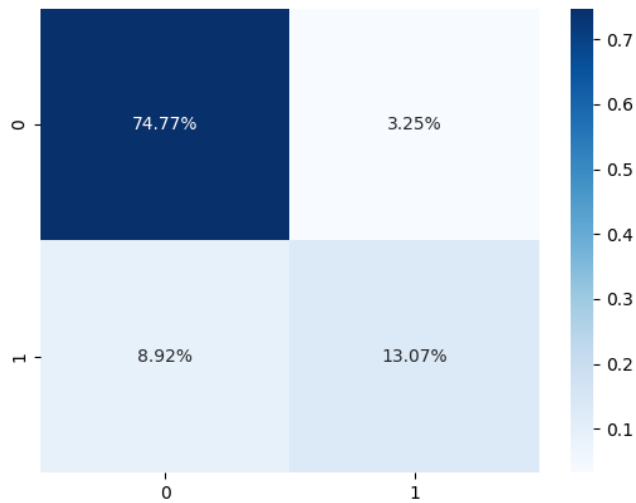
This is the bar chart created by function bar plot from seaborn. I find the all the distinct product category in the dataset, group by the product category and count each of the product in the dataset in descending. As we can see from the graph, the most product people shopped on Amazon is apparel and the least is furniture.

## 2. Star counting graph(EDA)



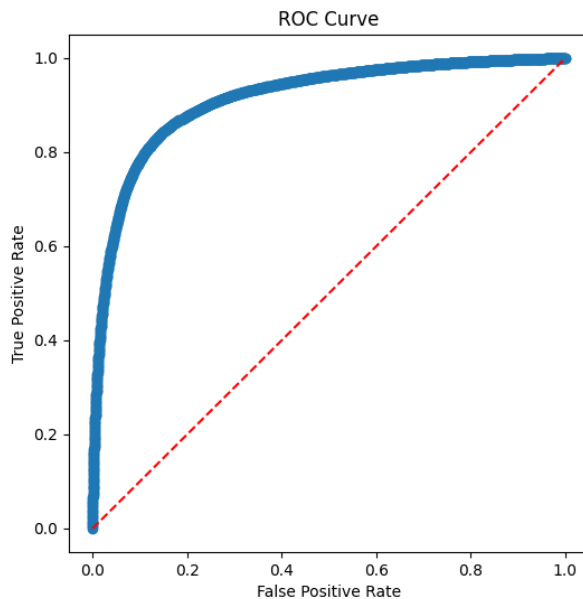
It is also the data exploratory graph which explored star rating range from 1 to 5. Obviously, we can find out that most of the reviews are 5 stars. Rating from 1 to 4 stars were relatively low in this dataset, then we can tell the quality of the products and service from Amazon are good and satisfied most of the customers.

### 3. Confusion matrix (Model performance)



The confusion matrix is to measure the classification model performance. The 0 here means positive and 1 is negative. As we can see that the true positive is 74.77 percent and true negative was 12.07 percent. They account for around 88 percent of the result, which means the model can correctly classify most of the customers attitude from their reviews.

#### 4. ROC curve (Model performance)



The ROC curve is created by false positive rate and true positive rate. As we can see our roc curve apparently close to top-left corner, which means we have a good model here and we also have larger area under the curve(AUC). We calculate the AUC by following code:

```
evaluator = BinaryClassificationEvaluator(metricName="areaUnderROC")
```

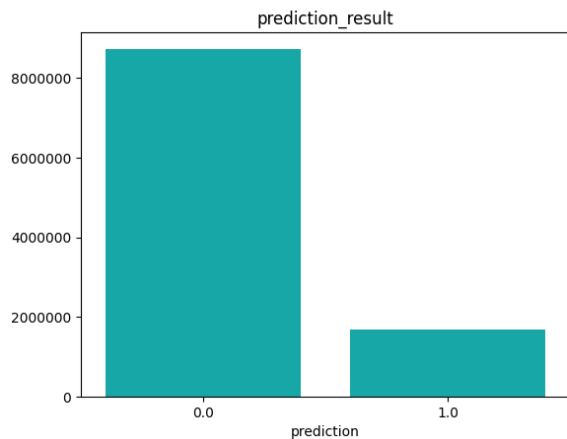
```
auc = evaluator.evaluate(predictions)
```

```
>>> print('AUC:', auc)
AUC: 0.9127619073159993
>>> █
```

Since our AUC is between 0.90 to 1, I believe our model perform pretty well in this project.



## 5. Prediction result graph



This graph was the result of the prediction, which obviously can see most of the prediction was positive reviews. And this is what we can expect from the star counting graph since most of the reviews are 5 stars and it would refer that large fraction of result will be positive.

## Milestone 6: Summary and Conclusion

To sum up, we successfully classify the attitude from customers' reviews, which started from finding the dataset, automating the process to download data from Kaggle to AWS S3, and do exploratory data analysis which extract some statistics of dataset. Then we start to clean the dataset, create our machine learning pipeline and used grid search and cross validator to create multiple models to help us to find most fitted hyperparameters to fit the trainset to produce the prediction results. In the end, we have some model evaluation such as confusion matrix, accuracy, precision, f1 score, recall, AUC and ROC curve then we create visualization both from EDA and model performance part and get some insight through the graphs. In my opinion, dealing with text data always need text-preprocessing process to remove those not important words to increase the accuracy of the model. Besides, we should better use hyperparameter tuning like grid search and cross validator to get a more accurate prediction result.

In conclusion, most of the result with obvious attitude such as "worst", "nice", "good", "happy", "unfortunately", "sad" are easy to recognize by the model. And I found out that for those false positive and false negative results were because some of the reviews they complimented the products, but they also mentioned some drawbacks of products. It might be the future works for me to used other tools to work on specific cases.

# Milestone 7: GitHub URL

<https://github.com/shanelin0107/CIS-4130.git>

## Appendix A: Code for downloading data

Region name – \$us-east-2

Output format- \$json

```
aws s3api create-bucket --mybucket4130 --region us-east-2 --create-bucket-configuration \ LocationConstraint=us-east-2
```

```
pip3 install kaggle
```

```
mkdir .kaggle
```

```
nano .kaggle/kaggle.json
```

```
chmod 600 .kaggle/kaggle.json
```

```
kaggle datasets list
```

```
nano ~/.local/lib/python3.7/sitepackages/kaggle/api/kaggle_api_extended.py
```

```
#change to these two lines
```

```
if not os.path.exists(outpath) and outpath != "-":
```

```
with open(outfile, 'wb') if outpath != "-" else os.fdopen(sys.stdout.fileno(), 'wb',  
closefd=False) as out:
```

```
kaggle datasets list
```

```
kaggle datasets download
```

```
kaggle datasets download --quiet -d cynthiarempel/amazon-us-customer-reviews-  
dataset-p - | aws s3 cp - s3://mybucket4130/amazonreview.zip
```

```
#Unzip the file
```

```
import zipfile
```

```
import boto3
```

```
from io import BytesIO
```

```
bucket="mybucket4130"
```

```
zipfile_to_unzip="amazonreview.zip"
```

```
s3_client = boto3.client('s3', use_ssl=False)
```

```

s3_resource = boto3.resource('s3')

zip_obj = s3_resource.Object(bucket_name=bucket, key=zipfile_to_unzip)
buffer = BytesIO(zip_obj.get()["Body"].read())
z = zipfile.ZipFile(buffer)
# Loop through all of the files contained in the Zip archive
for filename in z.namelist():
    print('Working on ' + filename)
    # Unzip the file and write it back to S3 in the same bucket
    s3_resource.meta.client.upload_fileobj(z.open(filename),
        Bucket=bucket, Key= f'{filename}')

```

## Appendix B: Carrying out descriptive statistics

### #data cleaning and statistics of data

```

from pyspark.sql.functions import col, isnan, when, count, udf, to_date, year, month,
date_format, size, split

```

```

#read the data from s3 bucket
df=spark.read.csv('s3n://mybucket4130/final_amz.csv',header=True)

```

```

#print all columns
df.columns

```

```

#print the summary of the dataframe
df.summary().show()

```

```

#check null values of star_rating and review_body column
df.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in
["star_rating", "review_body"]]).show()

```

```

#count how many records in the dataframe
df.count()

```

```

#drop null values
df = df.na.drop(subset=["star_rating", "review_body"])

```

```

#define function to drop the emoji from customer reviews
def ascii_only(mystring):
    if mystring:

```

```

        return mystring.encode('ascii', 'ignore').decode('ascii')
    else:
        return None
# assign function to udf
ascii_udf = udf(ascii_only)

#applied function to review body
df = df.withColumn("clean_review_body", ascii_udf('review_body'))

#save the cleaned data to csv
output_file_path="s3://mybucket4130/ cleaned_data.csv"
df.write.options(header='True', delimiter=',').csv(output_file_path)

```

## Appendix C: ML pipeline (cleaning, feature extraction, model building)

```

#import all library we need
Import io
import pandas as pd
import s3fs
import boto3
import matplotlib.pyplot as plt
import seaborn as sns
from pyspark.sql.functions import col, isnan, when, count, udf, to_date, year, month,
date_format, size, split
from pyspark.ml.stat import Correlation
from pyspark.ml.feature import VectorAssembler
from pyspark.sql.types import IntegerType
from pyspark.ml.feature import HashingTF, IDF, Tokenizer, StringIndexer
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from sklearn.metrics import confusion_matrix
from pyspark.ml.feature import StopWordsRemover
# read the clean data

df=spark.read.csv('s3n://mybucket4130/cleaned_data.csv',sep='\t', header=True,
inferSchema=True)

#drop the column we don't need

```

```
df=df.drop('_c0', 'marketplace', 'customer_id', 'review_id', 'product_id', 'product_parent',
'product_title', 'product_category','helpful_votes', 'total_votes', 'vine', 'verified_purchase',
'review_headline','review_body','review_date')
```

```
# check the null values in dataset
```

```
df.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in
["clean_review_body","star_rating"]]).show()
```

```
#remove any non numeric value in star rating
```

```
df = df.filter(~df.star_rating.rlike('\D+'))
```

```
# drop the rows with null values
```

```
df = df.na.drop(subset=["clean_review_body","star_rating"])
```

```
#convert the star_rating column from string to integer
```

```
df = df.withColumn("star_rating",df.star_rating.cast(IntegerType()))
```

```
#create a column to identify if the star rating >=3=1, otherwise is 0
```

```
df = df.withColumn("rating_converted", when(col("star_rating") > 3, 1).otherwise(0))
```

```
#split the dataset
```

```
train_set, test_set = df.randomSplit([0.7, 0.3], seed = 2000)
```

```
#convert the sentence to token
```

```
tokenizer = Tokenizer(inputCol="clean_review_body", outputCol="clean_review_words")
```

```
#remove stopwords
```

```
remover = StopWordsRemover(inputCol="clean_review_words", outputCol="remove_stopword")
```

```
#convert words to vectors
```

```
hashtf = HashingTF(numFeatures=2**16, inputCol="remove_stopword", outputCol='tf')
```

```
#create inverse document frequency
```

```
idf = IDF(inputCol='tf', outputCol="features", minDocFreq=5)
```

```
#create indexer for the rating_converted column
```

```
label_stringIdx = StringIndexer(inputCol = "rating_converted", outputCol = "label")
```

```
#create model
lr = LogisticRegression(maxIter=100)

#create pipeline
pipeline = Pipeline(stages=[tokenizer,remover, hashtf, idf, label_stringIdx, lr])

# Create a grid to hold hyperparameters
grid = ParamGridBuilder()
grid = grid.addGrid(lr.regParam, [0.0, 1.0])
grid = grid.addGrid(lr.elasticNetParam, [0, 1])

# Build the parameter grid
grid = grid.build()

# How many models to be tested
print('Number of models to be tested: ', len(grid))

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

# Create the CrossValidator using the hyperparameter grid
cv = CrossValidator(estimator=pipeline,
                    estimatorParamMaps=grid,
                    evaluator=evaluator,
                    numFolds=3,
                    seed=789
                    )

# Train the models
cv = cv.fit(train_set)

# Test the predictions
predictions = cv.transform(test_set)
```

```

# Calculate AUC
evaluator = BinaryClassificationEvaluator(metricName="areaUnderROC")
auc = evaluator.evaluate(predictions)
print('AUC:', auc)

# Create the confusion matrix
predictions.groupby('label').pivot('prediction').count().fillna(0).show()
cm = predictions.groupby('label').pivot('prediction').count().fillna(0).collect()
def calculate_recall_precision(cm):
    fn = cm[0][1] # False Negative
    tn = cm[0][2] # True Negative
    tp = cm[1][1] # True Positive
    fp = cm[1][2] # false Positive
    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) )

```

## Appendix D: Visualization

**Save each graph through following code:**

```

img_data=io.BytesIO()
plt.savefig(img_data,format='png')
img_data.seek(0)

```

```

s3=s3fs.S3FileSystem(anon=False)
with s3.open('s3://mybucket4130/file_name.png','wb') as f:
    f.write(img_data.getbuffer())

```

**visualization 1: product category counting**

```

gg=df.groupby('product_category').count().sort('count',ascending=False).show()

```

```

pd=['Apparel','Toys','Sports','Shoes','Electronics','Pet Products','Office
Products','Grocery','Camera','Tools','Furniture']

sb=gg.filter(gg.product_category.isin(pd))

sb.groupby('product_category').count().sort('count',ascending=False).show()

qty=sb.groupby('product_category').count().sort('count',ascending=False).toPandas()

grh=sns.barplot(x=qty['product_category'],y=qty['count'],color='c').set(title='product_counting')

grh.set_xticklabels(grh.get_xticklabels(),rotation=50)

plt.figure(figsize = (11,11))

plt.ticklabel_format(style='plain',axis='y')

```

---

### **Visualization 2: star rating counting**

```

str=df.groupby('star_rating').count().sort('count',ascending=False).toPandas()

grh=sns.barplot(x=str['star_rating'],y=str['count'],color='c').set(title='star_counting')

grh.set_xticklabels(grh2.get_xticklabels(),rotation=0)

```

---

### **Visualization 3: confusion matrix**

```

y_true = predictions.select("label")
y_true = y_true.toPandas()
y_pred = predictions.select("prediction")
y_pred = y_pred.toPandas()
cnf_matrix = confusion_matrix(y_true, y_pred)
sns.heatmap(cf_matrix/np.sum(cf_matrix), annot=True, fmt='.2%', cmap='Blues')

```

---

### **Visualization 4:ROC curve**

```

# Look at the parameters for the best model that was evaluated from the grid
parammap = cv.bestModel.stages[5].extractParamMap()
for p, v in parammap.items():
    print(p, v)
# Grab the model from Stage 5 of the pipeline
mymodel = cv.bestModel.stages[5]

```



```
plt.figure(figsize=(5,5))
plt.plot(mymodel.summary.roc.select('FPR').collect(),
mymodel.summary.roc.select('TPR').collect())
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title("ROC Curve")
```

---

### **Visualization 5: prediction result**

```
qty=predictions.groupby('prediction').count().toPandas()
sns.barplot(x=qty['prediction'],y=qty['count'],color='c')
plt.ticklabel_format(style='plain',axis='y')
plt.title('prediction_result')
with s3.open('s3://mybucket4130/predict_result2.png','wb') as f:
    f.write(img_data.getbuffer())
```

### **Sources:**

1. <https://github.com/Kaggle/kaggle-api/issues/315>
2. [Sentiment Analysis with PySpark. One of the tools I'm deeply interested... | by Ricky Kim | Towards Data Science](#)
3. [Sentiment-Analysis-and-Text-Classification-Using-PySpark/Food Review.ipynb at master · shikha720/Sentiment-Analysis-and-Text-Classification-Using-PySpark · GitHub](#)
4. [Confusion Matrix Visualization. How to add a label and percentage to a... | by Dennis T | Medium](#)