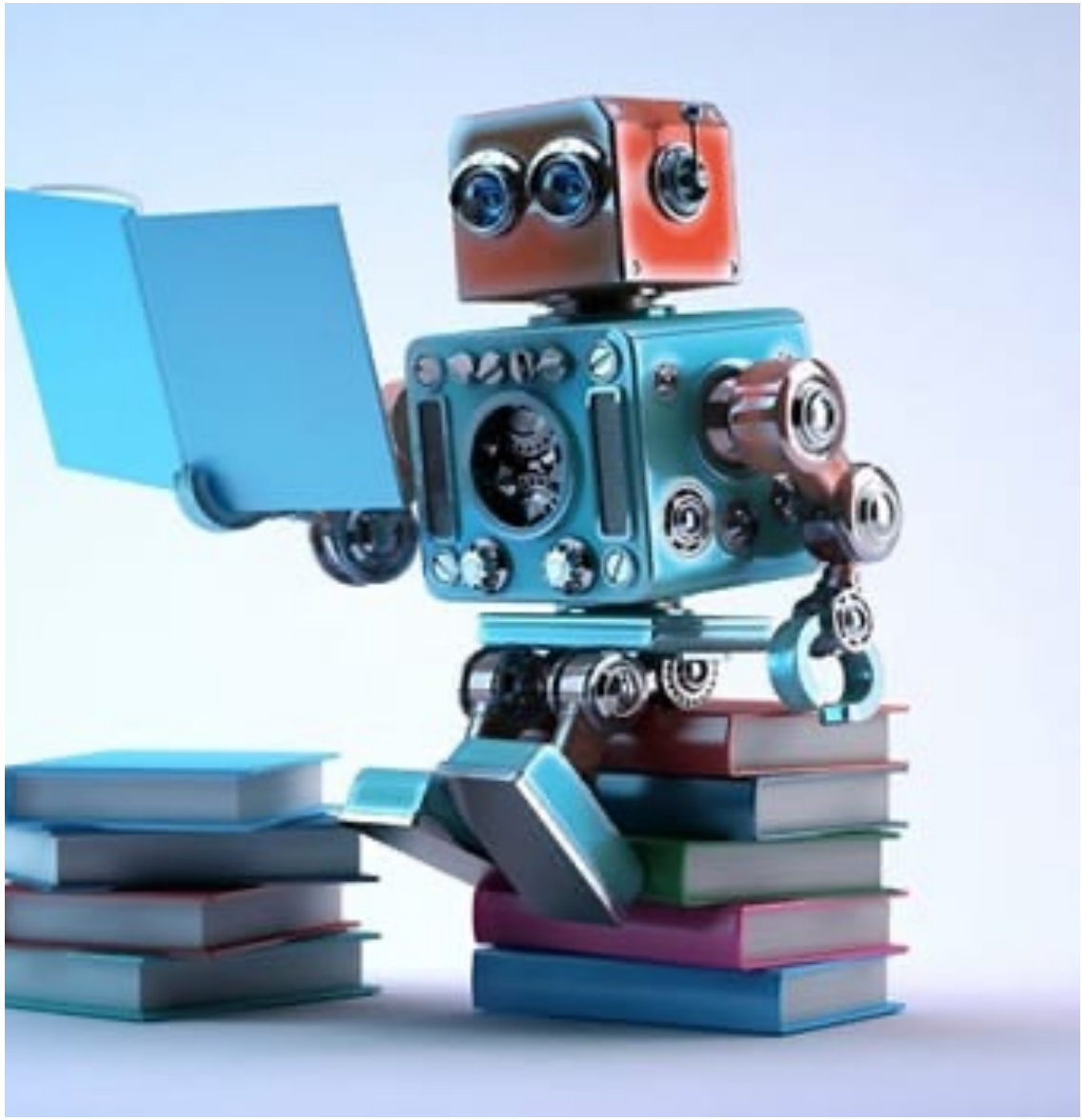
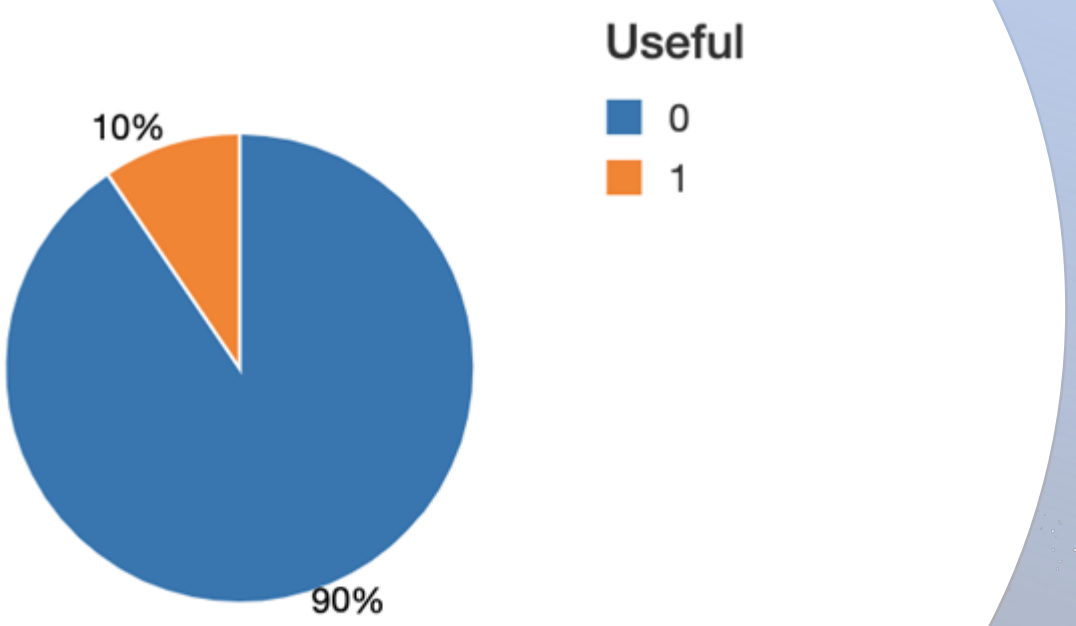



amazon

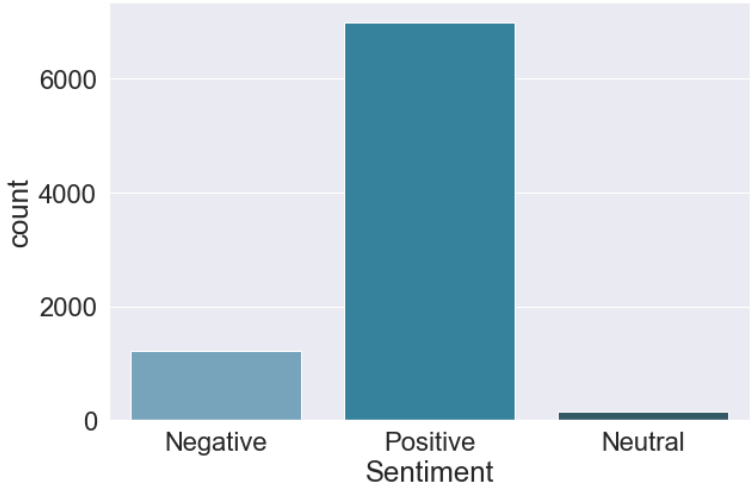


Objectives

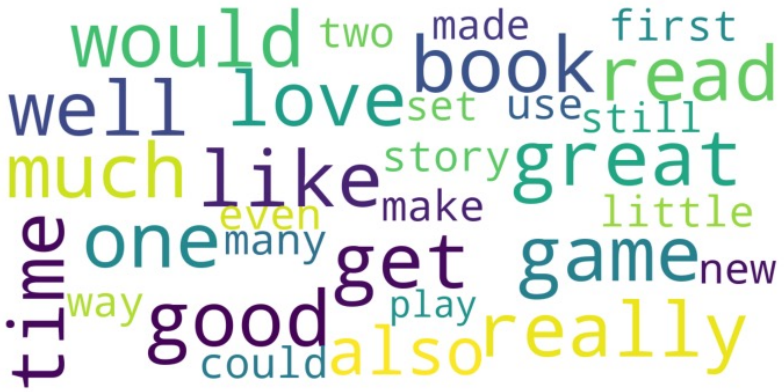
- Assessing factors through text analytics and language processing that impacts usefulness of an Amazon review
- Understand and recommend cloud computing features for productionalizing project

Table of Contents

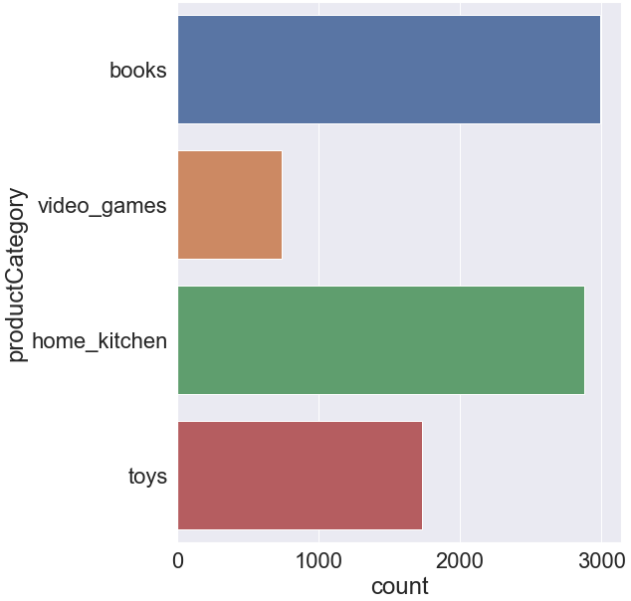
Review Sentiments



Common Words



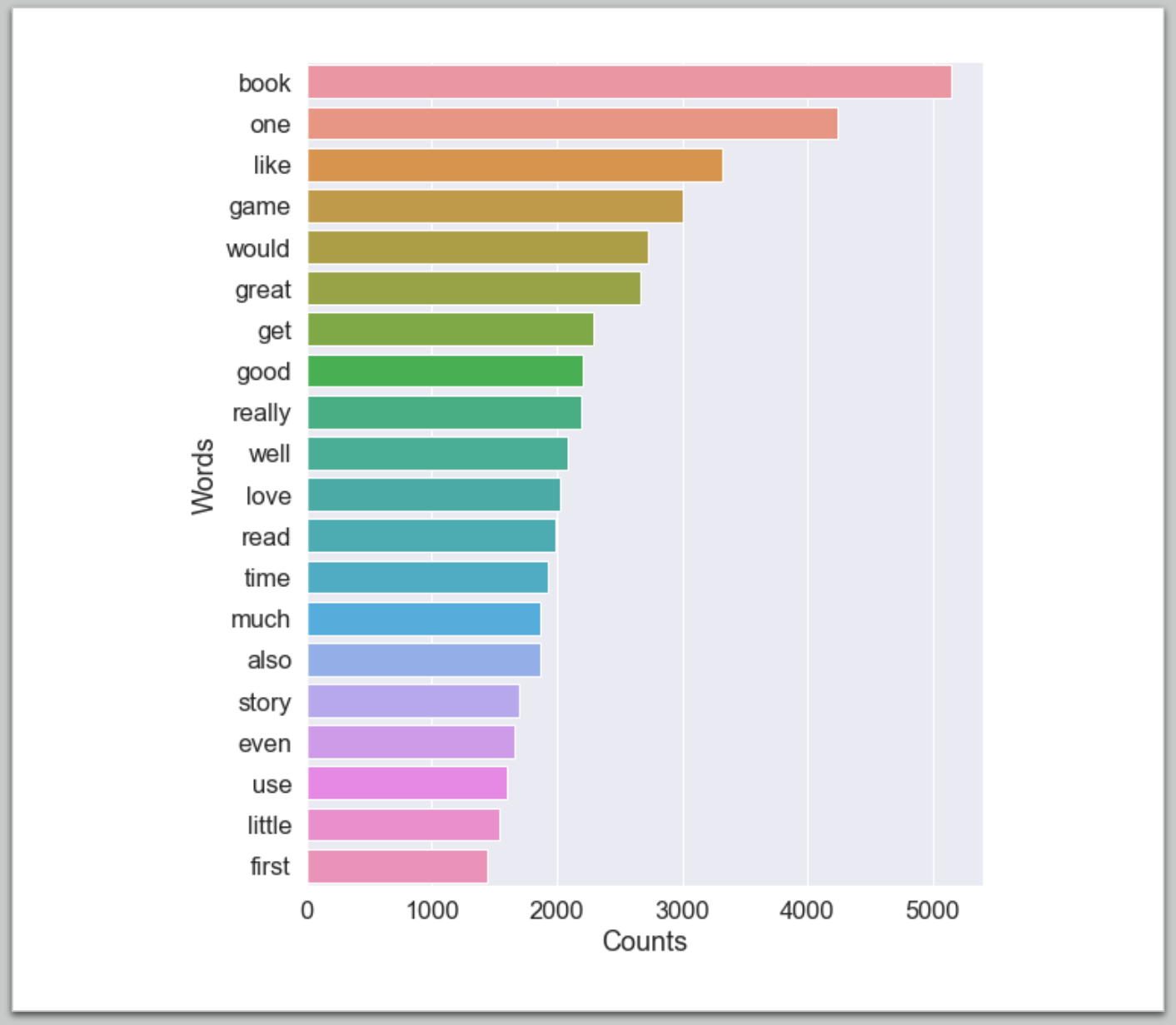
Product Categories



Trends

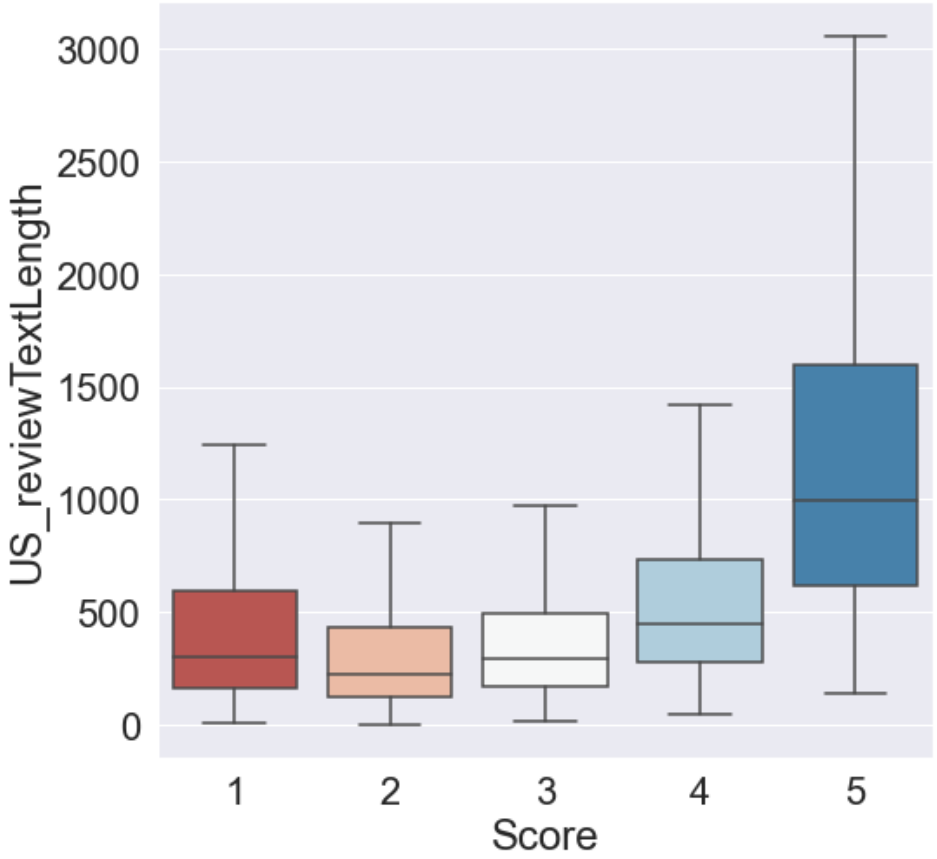
Text Analysis

- Utilized the sample data provided to perform text analysis
- Considered only useful reviews by filtering label column to only contain value 1
- NLP packages to filter for stop-words
- BOW determined by countVectorizer
- Displaying top 30 words from reviewText column



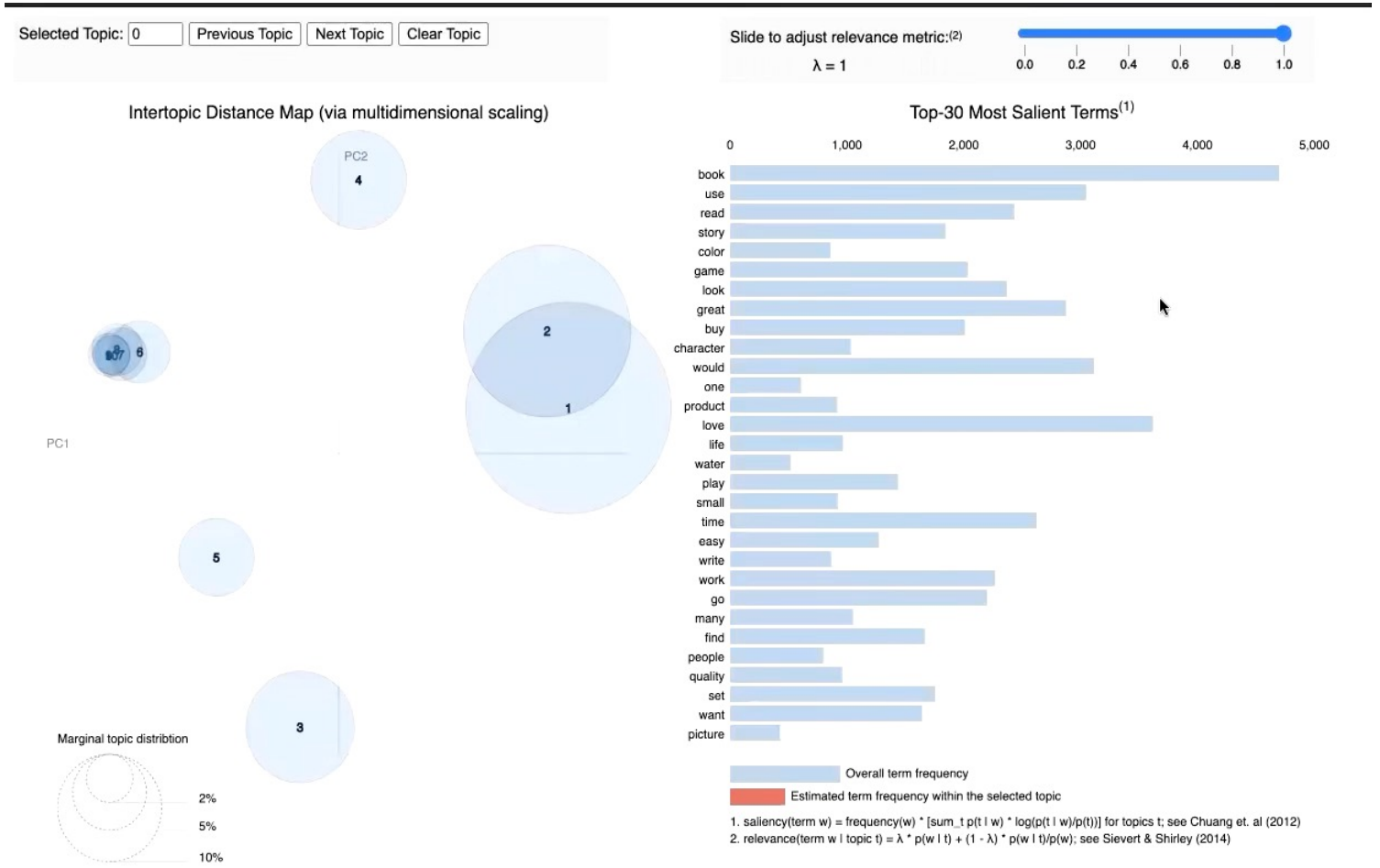
Text Analysis - Sentiments

- Categorized U*S_reviewTextSentiment* into 3 brackets
- Positive, negative and neutral (0s) values
- Overall, we can see "positive" sentiments in reviews
- Also analyzed sentiment versus length of review by splitting sentiment scores into 5 quantiles
- Scores over 0.96 have highest IQR and text length



Topics

- LDA using Gensim and pyLDAvis package to create an interactive map
- Selected n_topics = 10
- Created dictionary using lemmatized data
- Coherence score of 0.41



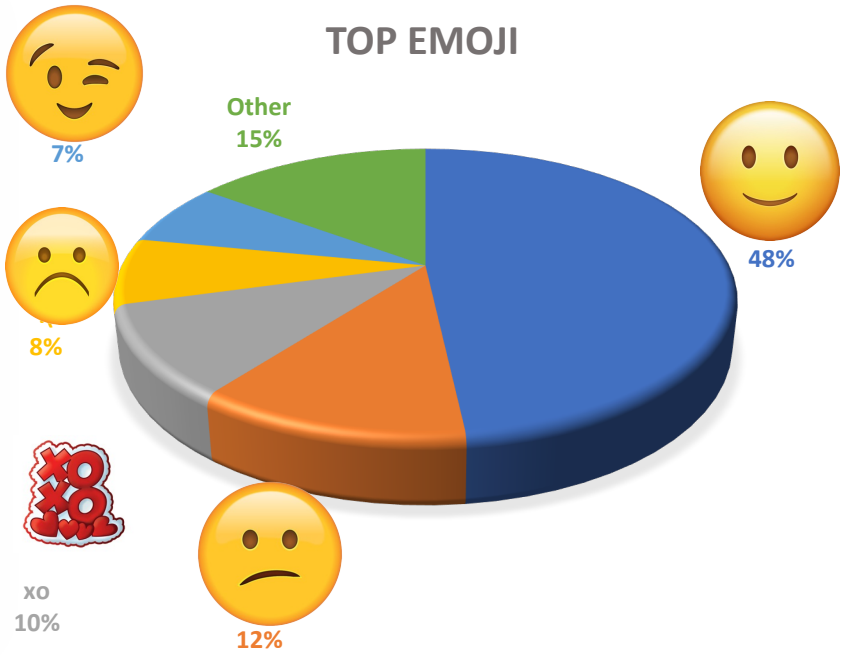
Data Cleaning

- Used Tokenization (\\W) to grab the words between special characters (. , ! ?)
- Removed stop words (i.e.: if, or, his, and)
- Lower case all words

Emoji Analysis

1. Created Emoji lexicon
2. Used RegExp to find smiley faces in reviewText and summary
3. If found, then populated corresponding emoji_type value into EMOJI field that explained the reviewer emotion
4. Result: Increase in AUC ~ 0.05 in Kaggle

emoji_str	emoji_type
):	sad_face
:(	sad_face
=(	sad_face
)=	sad_face
:)	smiley_face
(:	smiley_face
:d	delighted
=d	delighted
:o	surprised
:O	surprised
:0	surprised
xo	hugs_kisses
xoxo	hugs_kisses
<3	heart



Pipeline Features

1. (One Hot Encoding) productCategory category variable conversion into the form required by ML resulted as 3 new features

Consists of 2 features:

- Flesch Reading Ease (difficulty in understanding a para) FINAL RESULT
- Automated Readability Index (comprehensibility) PROCESS

Main Features

productCategoryOHE2_books	
productCategoryOHE2_home_kitchen	
productCategoryOHE2_toys	
verified	Verified that reviewer purchased the product
overall	Rating of the product
US_reviewTextLength	
US_reviewTextMetrics_0	
US_reviewTextMetrics_1	
US_reviewTextSentiment	
US_reviewAgeDays	

Product Category
books
home_kitchen
toys

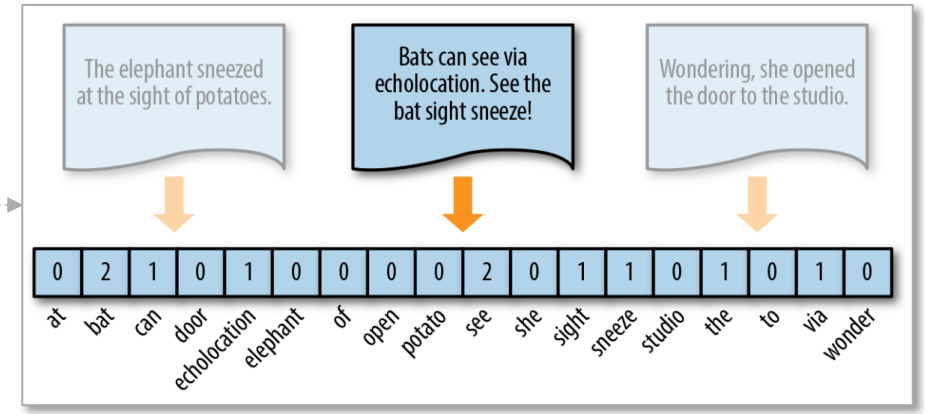
transpose

books	home_ kitchen	toys
0	1	0
1	0	0
0	0	1

2. Tokenization triggered 1,000 new features (words count)

Text Processing Features

idfFeatures_0
idfFeatures_1
idfFeatures_2
idfFeatures_3

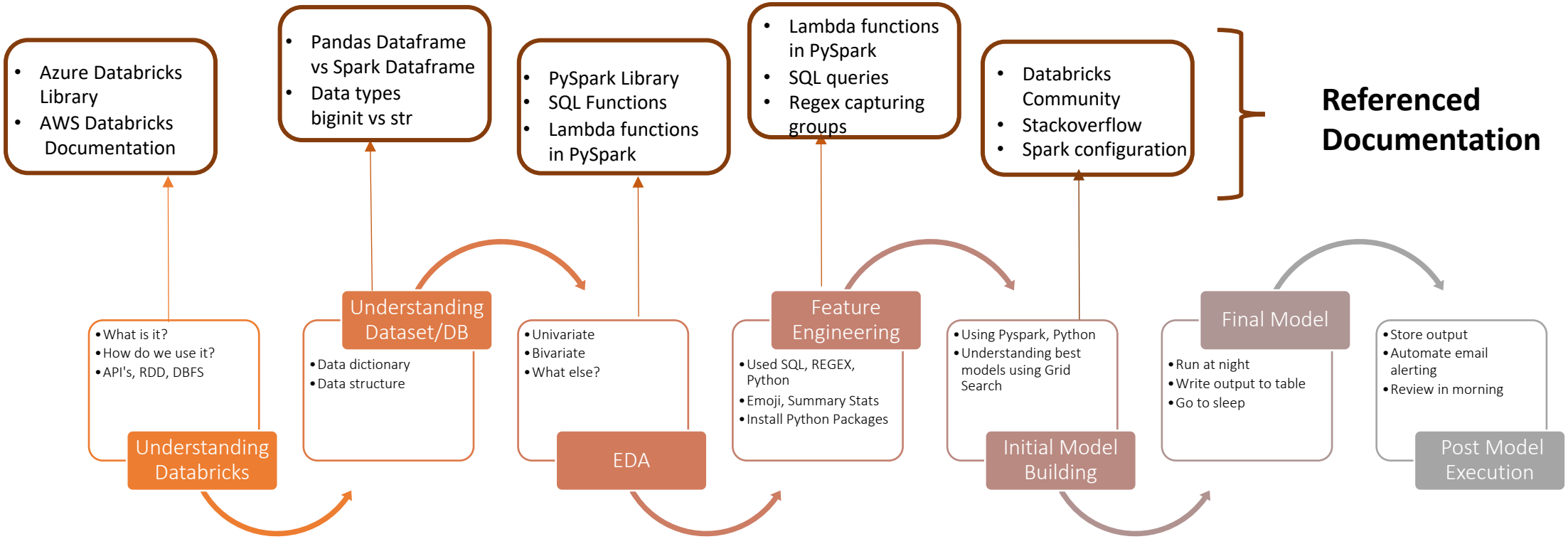


https://www.oreilly.com/library/view/applied-text-analysis/9781491963036/ch04.html

Databricks Learnings

- Much more powerful than local PC or single server
- Scales automatically but limited by Spark configurations and cluster settings
- Better than Jupiter Notebooks due to real-time changes and support for multiple languages
- Quick and easy dashboard creation. We can change dashboards in real-time
- Notebook version control for tracking and auditing
- ML flow logs are extremely useful in analyzing model parameters (limited by Databricks permissions)

Modeling Journey



Model Selection

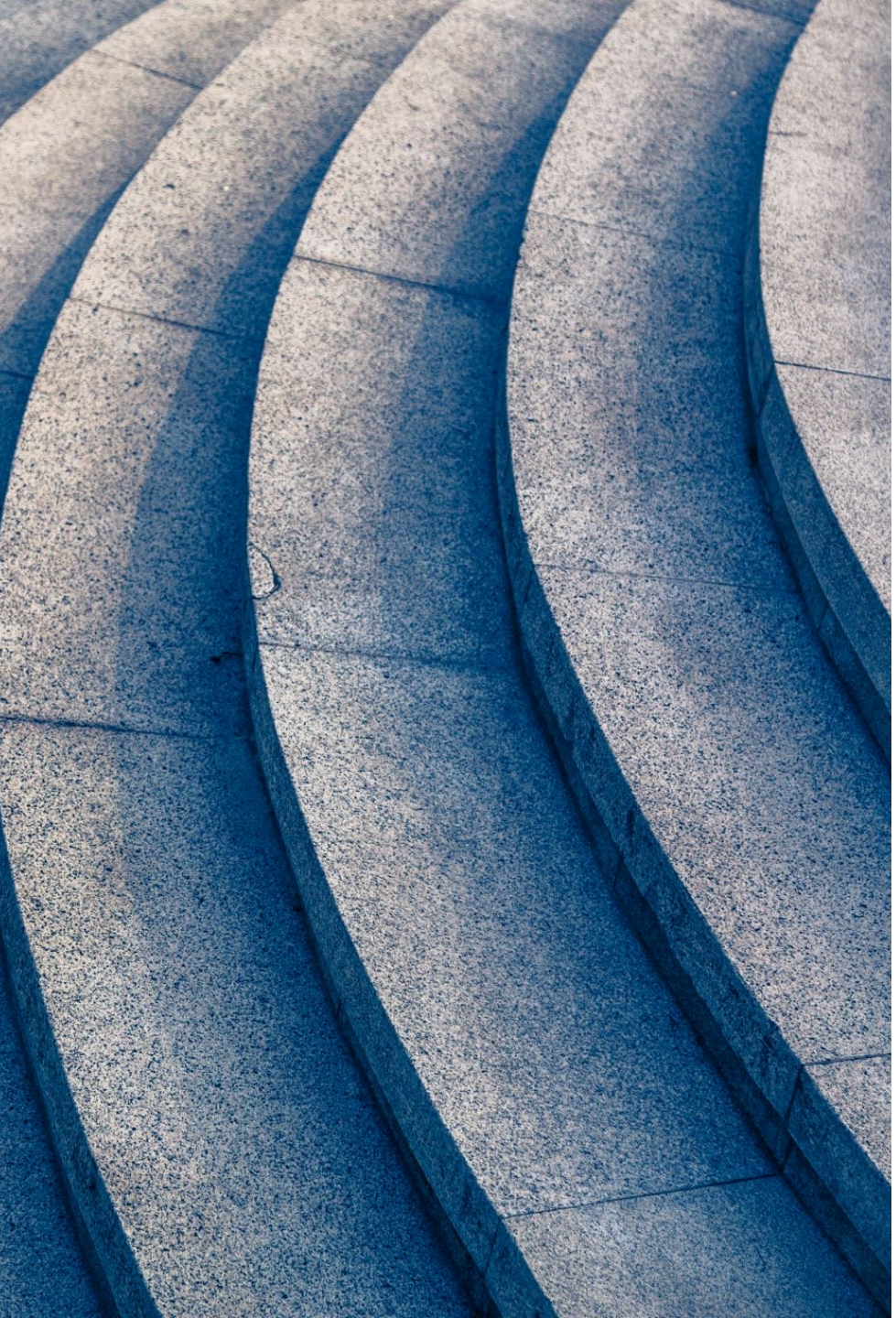
- Experimented with 5 modeling techniques
- Train data used 80/20 split & ranking system by Uncle Steve
- Tokenizer, StopWordRemover, TF, NGrams word convertor applied to training set
- Explored LR, DT and RF on test set for Kaggle
- RF MaxDepth of 250
- Selected RF based on best AUC value

Training Data Performance

Model	AUC	F1 Score
Logistic Regression	0.9066	0.8674
Naïve Bayes	0.8397	0.8506
Decision Tree	0.8857	0.8685
Random Forest	0.9060	0.8662
GBT Classifier	0.8692	0.8638

Top Kaggle Submissions

Model	AUC
Logistic Regression	0.81470
Decision Tree	0.82608
Random Forest	**0.82639**



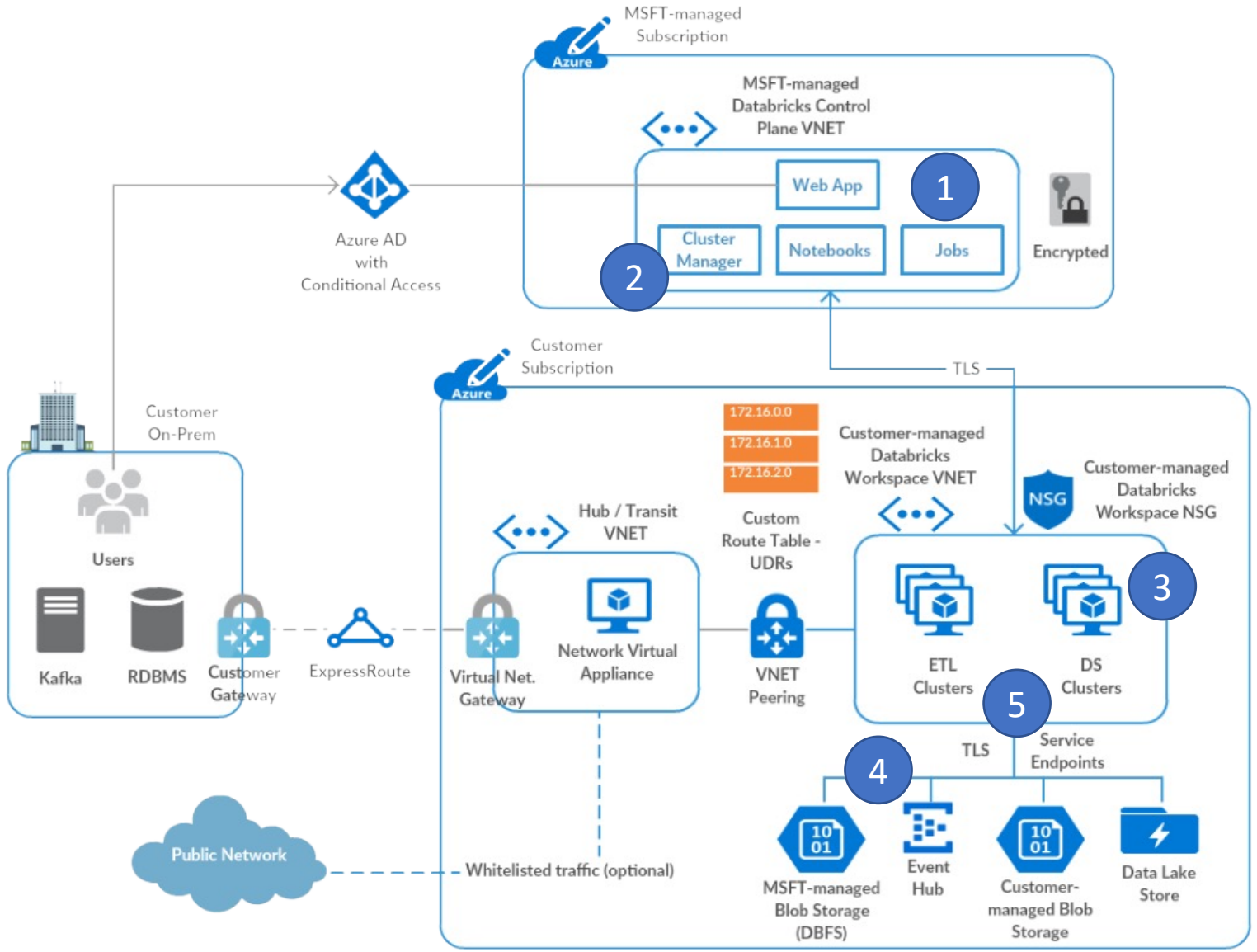
Next Steps

- Broken down into two streams
- Stream 1:
  - Additional data exploration, hunting
  - What else can we do with our script to extract features from the webpage
  - Can we pay amazon to provide additional features?
  - Continue to discover/create dynamic text/language-based features to understand how trends are changing
- Stream 2:
  - Covered in next slide

Thinking About Productionalizing & Optimizing ML in Databricks

Areas of performance improvements in Databricks and Azure

1. Increase jobs & tasks sent to worker nodes
2. Increase cores available for each task
3. Continuously monitor cluster cpu/mem performance
4. Integration with cheaper Azure storage
5. Azure rest API connectivity for cost/storage and service monitoring and management



Total Cost – EOD Feb 16th

VM Costs * Hrs

+

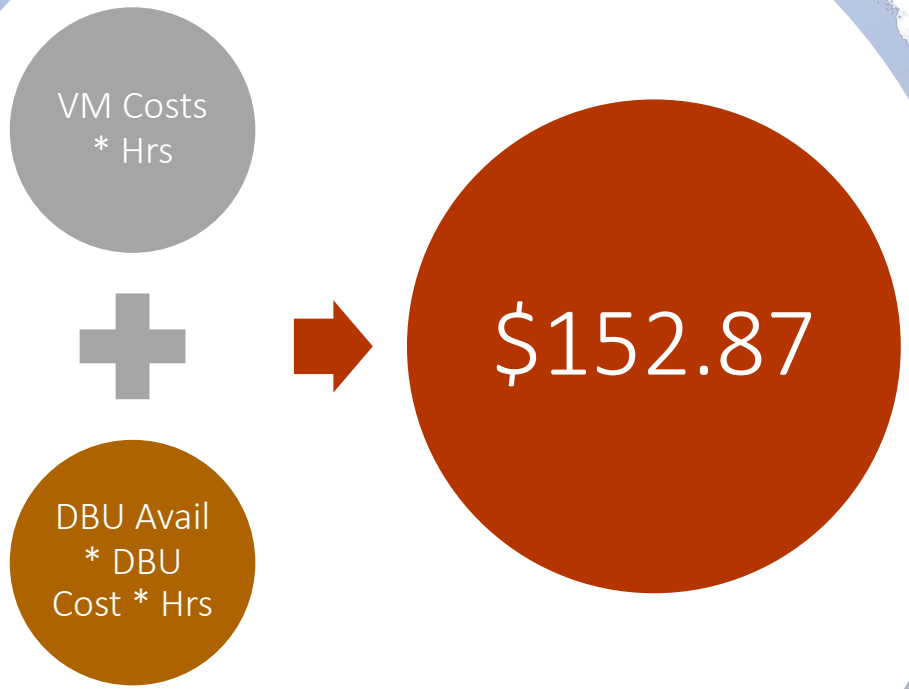
DBU Avail * DBU Cost * Hrs

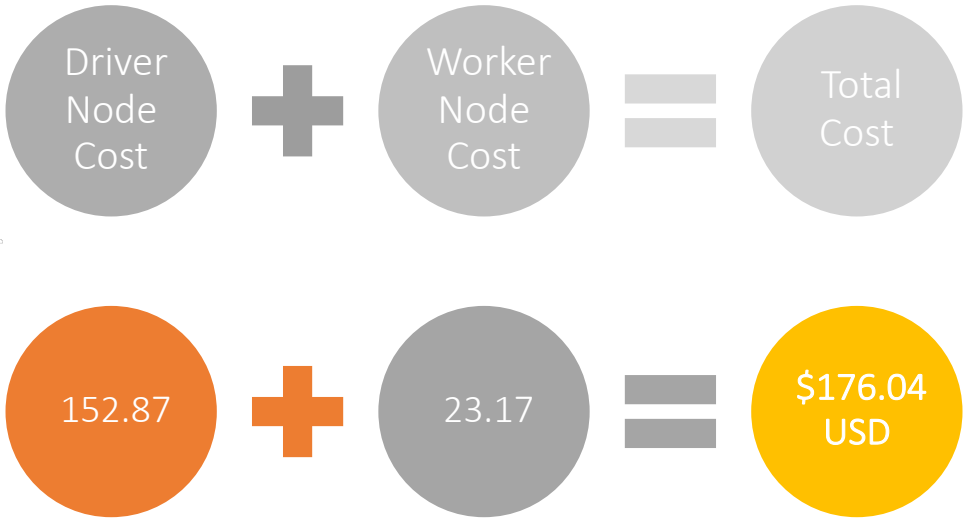
→ $152.87

Universal Cost Factors:

1. Total active time ~ 198.53 Hours Standard Tier Databrick Unit (DBU)
2. Active time is time difference between starting and terminating time
3. Assumed resizing from 1 to 4 happens immediately for simlicity
4. 1 Databrick Unit is utilized 100% of the time when the VM is 'starting'
5. Cluster type D12S

Total Cost – EOD Feb 16th



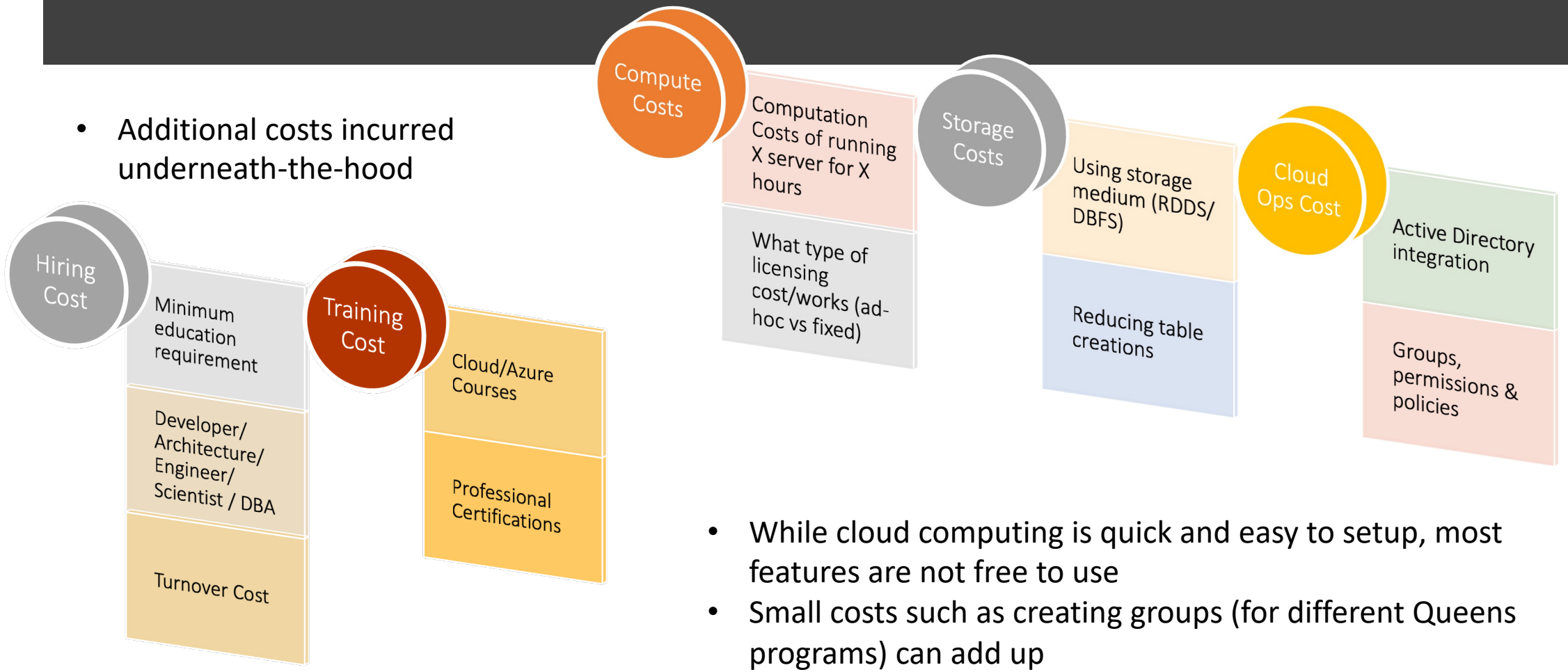


Universal Cost Factors:

1. Total active time ~ 198.53 Hours Standard Tier Databrick Unit (DBU)
2. Active time is time difference between starting and terminating time
3. Assumed resizing from 1 to 4 happens immediately for simplicity (total of 10 hrs)
4. 1 Databrick Unit is utilized 100% of the time when the VM is 'starting'
5. Cluster type standard D12S – pay as you go services

Additional Costs Underneath The Hood

- Additional costs incurred underneath-the-hood



- While cloud computing is quick and easy to setup, most features are not free to use
- Small costs such as creating groups (for different Queens programs) can add up
- Be prepared to spend more over time with cloud computing

Appendices

Azure AD Costs

Pricing details

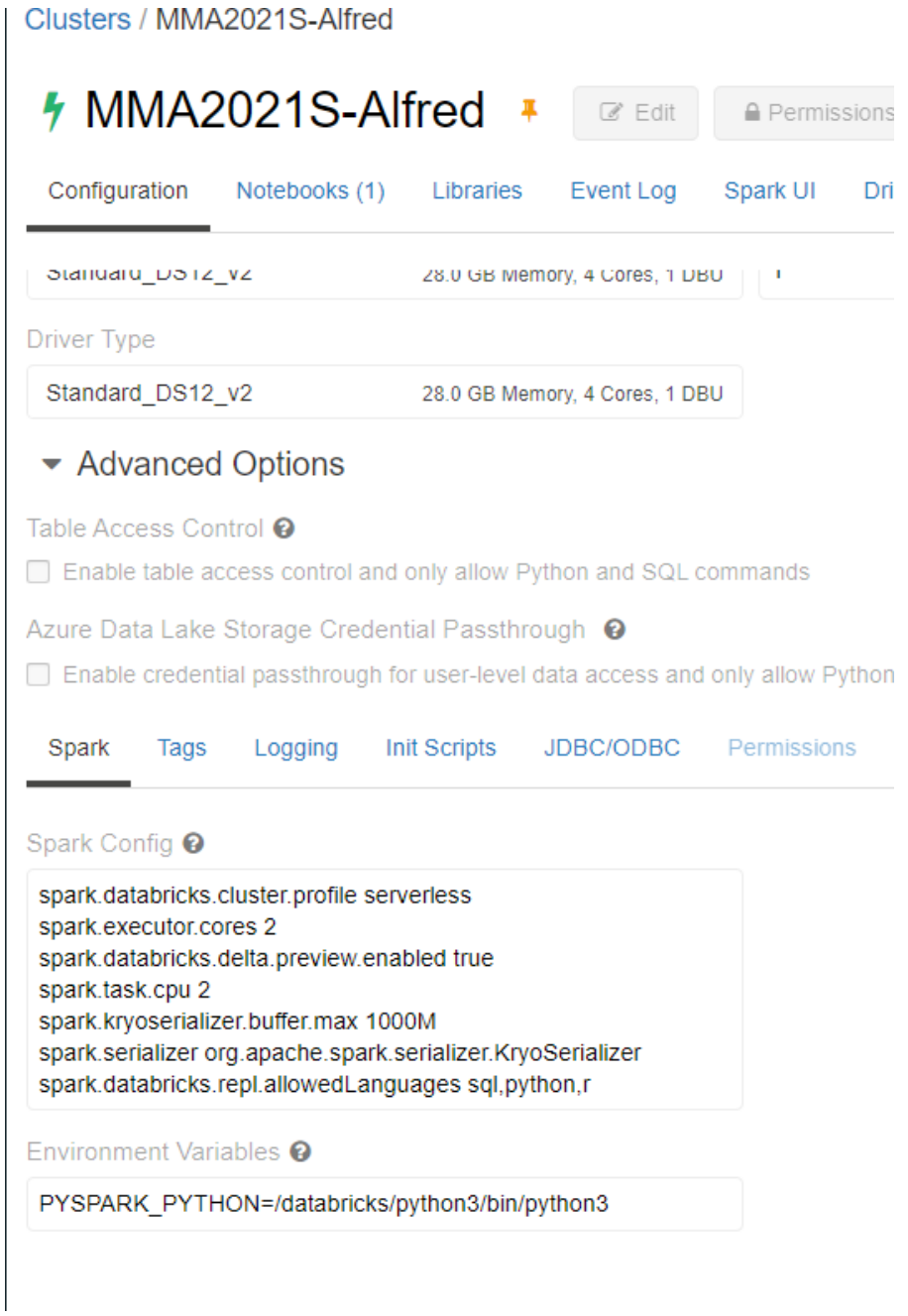
Azure Active Directory Domain Services usage is charged per hour, based on the SKU selected by the tenant owner. Azure Active Directory is available in User Forest and Resource Forest (currently in preview). While in preview, Resource Forest pricing includes a pricing discount. Prices listed in the table below include the preview discount.

	STANDARD	ENTERPRISE	PREMIUM
AAD DS Core Service			
Suggested auth load (peak, per hour)[1]	0 to 3,000	3,000 to 10,000	10,000 to 70,000
Suggested object count[2]	0 to 25,000	25,000 to 100,000	100,000 to 500,000
Backup frequency	Every 5 days	Every 3 days	Daily[3]
Resource Forest Trusts	N/A	5	10
Instances			
User Forest[4]	$0.20/hour/set	$0.52/hour/set	$2.05/hour/set
Resource Forest (Preview)[4]	N/A	$0.26/hour	$1.03/hour

ML Flow Logs

Filter based on any parameter from your defined grid search OR algorithm

Spark Cluster Configs

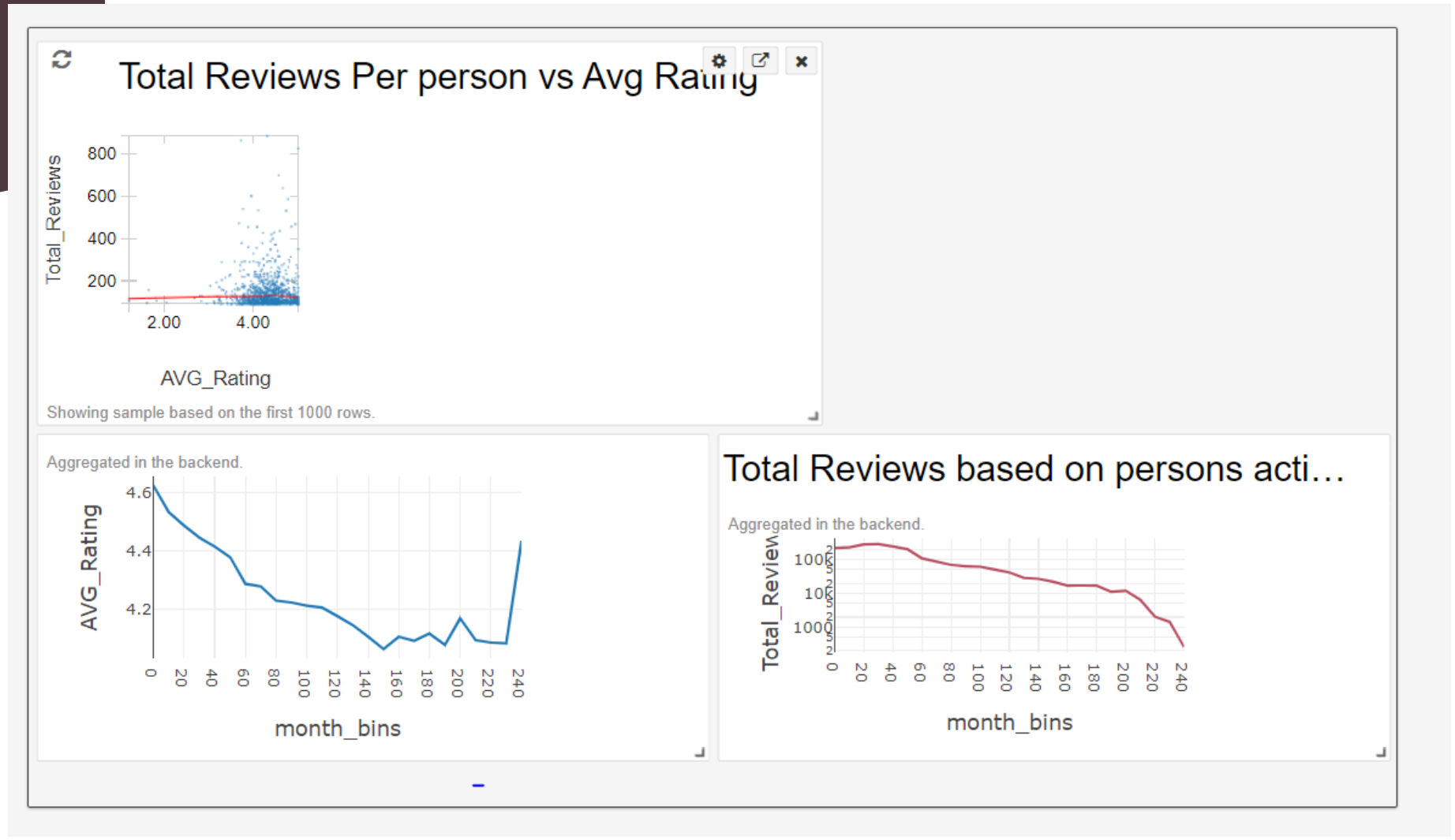


Custom Libraries

Uninstall | Install New

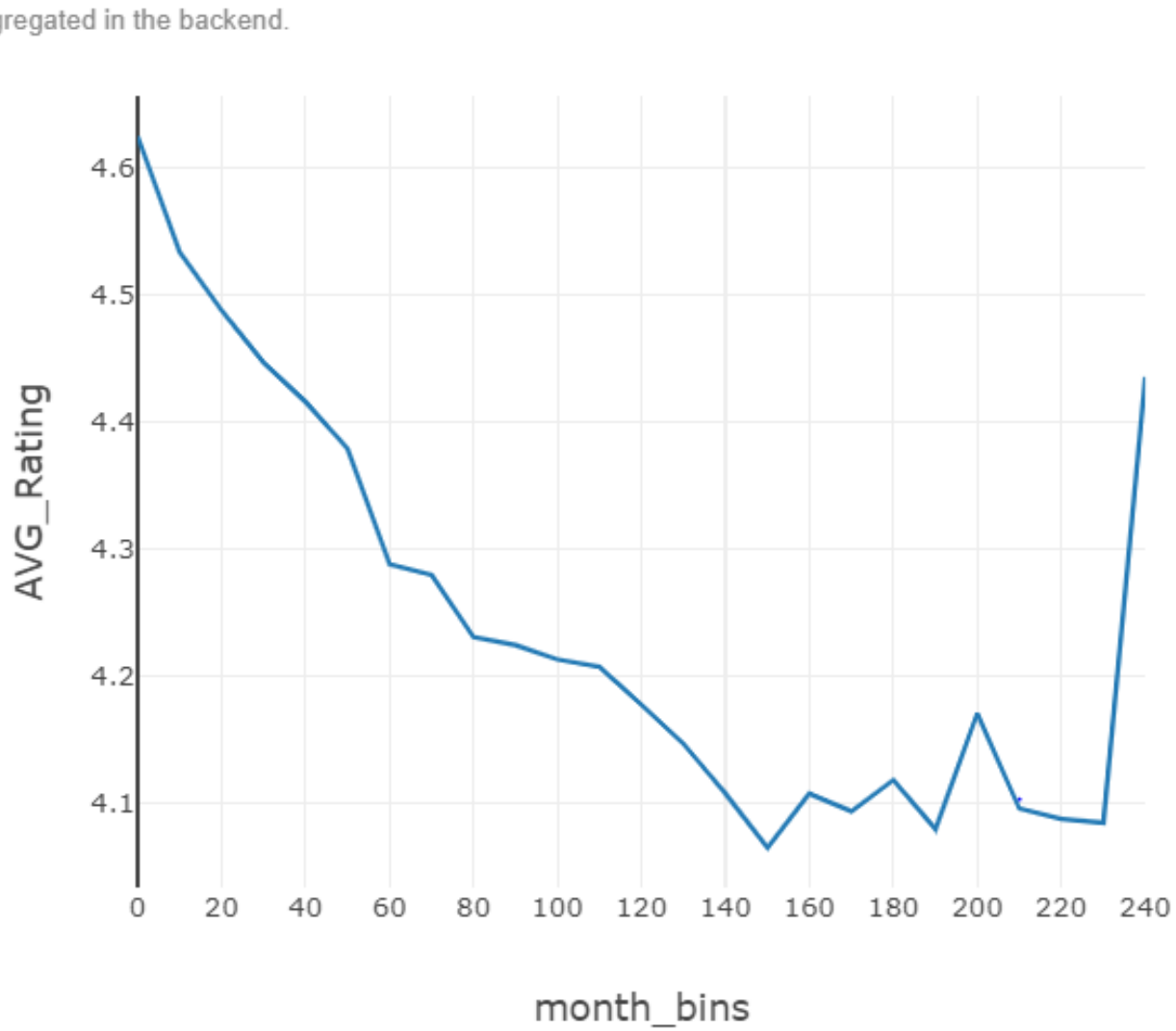
Name	Type	Status
vaderSentiment	PyPI	Installed
nltk	PyPI	Installed
sklearn	PyPI	Installed
pandas	PyPI	Installed
unidecode	PyPI	Installed
textblob	PyPI	Installed
textstat	PyPI	Installed
com.johnsnowlabs.nlp:spark-nlp_2.11:2.7.2	Maven	Installed
spark-nlp==2.7.2	PyPI	Installed

Persons Review Analysis



Total Reviews by Age (Months)

Months Active vs Average Review



Regex for Emoji

```
df = df.withColumn('emoji',regexp_extract('reviewText','(\:\\)|[\:\=\;][pP]|(?:\=\\))|(?:\;\\))|\<3|\:(?:b|B)|[\:\=\;]d|d[\:\=\;]|\:\/|\:\\(|\\)\:|\=\\(|\\)\=|\:(?:o|O|0)|[bB]\\)|\\)[\=\:\;]|xoxo|xo|\:\-\\\|[\>\<][\:\=\;]|[\:\=\;][\>\<]|[\:\=\;]\@|\@[\:\=\;]|x\\(|\\)x|[\:\=\;][sS$]|[\:\=\;]\-[Ss$]|[sS$][\:\=\;]|[Ss$]\-[\:\=\;]|[\:\=\;]\$|[\:\=\;]\@|x\@|[\=\:\;][sS$])',1))


df = df.withColumn("emoji_type", when(col("emoji") == ":)","smiley_face")\
                        .when(col("emoji") == "(:","smiley_face").when(col("emoji") == ":)","smiley_face").when(col("emoji") == "(=","smiley_face").when(col("emoji") == "=)","smiley_face")\
                        .when(col("emoji") == ":(","sad_face").when(col("emoji") == "):","sad_face").when(col("emoji") == "=(","sad_face").when(col("emoji") == ")=","sad_face")\
                        .when(col("emoji") == ":p","cheeky").when(col("emoji") == "=p","cheeky").when(col("emoji") == ";p","cheeky")\
                        .when(col("emoji") == ':\\','confused").when(col("emoji") == '\\:',"confused").when(col("emoji") == "=\\","confused").when(col("emoji") == "\\=","confused").when(col("emoji") ==
':/',"confused").when(col("emoji") == '/:',"confused").when(col("emoji") == "=/","confused").when(col("emoji") == "/=","confused").when(col("emoji") == ";-\\","confused").when(col("emoji") == ":-\\","confused")\
                        .when(col("emoji") == ";)","wink").when(col("emoji") == "(;","wink")\
                        .when(col("emoji") == "xo","sad_face").when(col("emoji") == "xoxo","hugs_kisses")\
                        .when(col("emoji") == "x(","angry").when(col("emoji") == ")x","angry").when(col("emoji") == "=@","angry").when(col("emoji") == "@=","angry").when(col("emoji") == ":s","angry").when(col("emoji") ==
"s:","angry")\
                        .when(col("emoji") == "=<","smug").when(col("emoji") == ":<","smug").when(col("emoji") == ";<","smug").when(col("emoji") == "d:","smug")\
                        .when(col("emoji") == ">;","smug").when(col("emoji") == ">=","smug").when(col("emoji") == ">:","smug").when(col("emoji") == "d:","smug")\
                        .when(col("emoji") == "d=","smug").when(col("emoji") == "d;","smug")\
                        .when(col("emoji") == "=d","delighted").when(col("emoji") == ":d","delighted").when(col("emoji") == "b)","sunglasses")\
                        .when(col("emoji") == ":0","surprised").when(col("emoji") == "=0","surprised").when(col("emoji") == ":o","surprised").when(col("emoji") == "=o","surprised")\
                        .when(col("emoji") == ":s","sick").when(col("emoji") == ":-s","sick").when(col("emoji") == "=-s","sick").when(col("emoji") == ";s","sick").when(col("emoji") == "=s","sick").when(col("emoji") ==
":$","sick").when(col("emoji") == ";$","sick").when(col("emoji") == "=$","sick")\
                        .when(col("emoji") == "s:","sick").when(col("emoji") == "s-:","sick").when(col("emoji") == "s-=","sick").when(col("emoji") == "s;","sick").when(col("emoji") == "s=","sick").when(col("emoji") ==
"$:","sick").when(col("emoji") == "$;","sick").when(col("emoji") == "$=","sick")\
                        .when(col("emoji") == "<3","heart").when(col("emoji") == "<\\3","heart")\
                        .otherwise("None"))
```

Code for pipeline features

```
#check if we are at the proper pipeline stage (should be returning Linear Regression)
pipelineFit.stages[-1]
#------------------------------------------------

#get the feature coefficients
lrm = pipelineFit.stages[-1]
transformed =  pipelineFit.transform(trainingData)

#Extracting ML attr
from itertools import chain

attrs = sorted(
    (attr["idx"],attr["name"]) for attr in (chain(*transformed
        .schema[lrm.summary.featuresCol]

[(name, lrm.coefficients[idx]) for idx, name in attrs]
```

New columns for Sentiment of reviews

```
conditions = [(df['US_reviewTextSentiment'] < 0),
        (df['US_reviewTextSentiment'] == 0),
        (df['US_reviewTextSentiment'] > 0)]

values = ['Negative', 'Neutral', 'Positive']

df['Sentiment'] = np.select(conditions, values)

conds = [df.US_reviewTextSentiment.between(-1,0.27446),
      df.US_reviewTextSentiment.between(0.27446,0.77258),
      df.US_reviewTextSentiment.between(0.77258,0.91070),
      df.US_reviewTextSentiment.between(0.91070,0.96780),
      df.US_reviewTextSentiment.between(0.96780,1)]

choices = [1,2,3,4,5]

df['Score'] = np.select(conds,choices)
```

LDA Results - Topics

[(0,
'0.019*"learn" + 0.018*"brand" + 0.011*"people" + 0.011*"book" + 0.011*"result" + 0.011*"understand" + 0.010*"many" + 0.009*"question" + 0.009*"history" + 0.008*"disappointed"'),
(1,
'0.023*"great" + 0.022*"look" + 0.020*"game" + 0.016*"buy" + 0.015*"set" + 0.014*"play" + 0.013*"easy" + 0.012*"get" + 0.011*"well" + 0.011*"good"'),
(2,
'0.064*"color" + 0.032*"picture" + 0.030*"soft" + 0.019*"cook" + 0.016*"black" + 0.016*"pan" + 0.015*"replace" + 0.014*"push" + 0.014*"edge" + 0.014*"pot"'),
(3,
'0.072*"use" + 0.028*"product" + 0.019*"small" + 0.015*"size" + 0.015*"plastic" + 0.015*"bed" + 0.014*"bag" + 0.013*"quality" + 0.013*"buy" + 0.012*"hold"'),
(4,
'0.105*"book" + 0.057*"read" + 0.043*"story" + 0.024*"character" + 0.023*"life" + 0.019*"write" + 0.016*"author" + 0.015*"enjoy" + 0.012*"world" + 0.011*"live"'),
(5,
'0.022*"love" + 0.020*"would" + 0.019*"make" + 0.016*"time" + 0.014*"go" + 0.013*"well" + 0.013*"get" + 0.013*"good" + 0.013*"really" + 0.011*"work"'),
(6,
'0.057*"water" + 0.027*"extra" + 0.021*"baby" + 0.020*"paint" + 0.017*"lock" + 0.015*"tear" + 0.014*"attach" + 0.014*"town" + 0.013*"seller" + 0.012*"mess"'),
(7,
'0.047*"s" + 0.046*"candle" + 0.036*"warm" + 0.022*"green" + 0.021*"recipe" + 0.021*"regular" + 0.020*"burn" + 0.020*"contact" + 0.019*"brother" + 0.019*"rod"'),
(8,
'0.065*"one" + 0.028*"heat" + 0.027*"super" + 0.026*"perfectly" + 0.021*"stuff" + 0.019*"inch" + 0.017*"apart" + 0.016*"thin" + 0.016*"nicely" + 0.016*"oil"'),
(9,
'0.050*"card" + 0.034*"company" + 0.023*"war" + 0.017*"safe" + 0.017*"sharpen" + 0.012*"knife" + 0.012*"corner" + 0.012*"danger" + 0.012*"count" + 0.012*"element"')]

Training Data - Modeling Results

```
######## Print classification_report
prediction_and_labels=prediction.select("label","prediction")
y_true = []
y_pred = []
for x in prediction_and_labels.collect():
    xx = list(x)
    try:
        tt = int(xx[1])
        pp = int(xx[0])
        y_true.append(tt)
        y_pred.append(pp)
    except:
        continue

target_names = ['neg 0', 'pos 1']
print (classification_report(y_true, y_pred, target_names=target_names))
return
```

Logistic Regression

```
average cross-validation accuracy = 0.8271482726427045
Accuracy in the test data = 0.9045209756490215
F1 score in the test data = 0.8695780948371234
              precision    recall  f1-score   support

       neg 0       0.99      0.91      0.95   1276882
       pos 1       0.05      0.36      0.08     15140

   micro avg       0.90      0.90      0.90   1292022
   macro avg       0.52      0.63      0.52   1292022
weighted avg       0.98      0.90      0.94   1292022
```

Training Data - Modeling Results (cont.)

NaïveBayes

```
average cross-validation accuracy = 0.7705930472911964
Accuracy in the test data = 0.8396958410924892
F1 score in the test data = 0.8505850314142294
              precision    recall  f1-score   support

       neg 0       0.89      0.93      0.91     11249
       pos 1       0.32      0.24      0.27      1639

   micro avg       0.84      0.84      0.84     12888
   macro avg       0.61      0.58      0.59     12888
weighted avg       0.82      0.84      0.83     12888
```

Decision Tree

```
average cross-validation accuracy = 0.8043610326405263
Accuracy in the test data = 0.8857076350093109
F1 score in the test data = 0.8685055601273716
              precision    recall  f1-score   support

       neg 0       0.96      0.92      0.94     12278
       pos 1       0.14      0.28      0.19       610

   micro avg       0.89      0.89      0.89     12888
   macro avg       0.55      0.60      0.56     12888
weighted avg       0.92      0.89      0.90     12888
```

Training Data - Modeling Results(cont.)

Random Forest

average cross-validation accuracy = 0.7997763316345727
Accuracy in the test data = 0.8691806331471136
F1 score in the test data = 0.8638430964418571

	precision	recall	f1-score	support
neg 0	0.94	0.92	0.93	11887
pos 1	0.21	0.26	0.23	1001
micro avg	0.87	0.87	0.87	12888
macro avg	0.58	0.59	0.58	12888
weighted avg	0.88	0.87	0.87	12888

GBT Classifier

average cross-validation accuracy = 0.8243238283496668
Accuracy in the test data = 0.9059590316573557
F1 score in the test data = 0.8662239107581887

	precision	recall	f1-score	support
neg 0	1.00	0.91	0.95	12819
pos 1	0.02	0.41	0.04	69
micro avg	0.91	0.91	0.91	12888
macro avg	0.51	0.66	0.50	12888
weighted avg	0.99	0.91	0.95	12888

Amazon Review

Presentation by Team Alfred