In [0]:

```
from pyspark.sql import SQLContext, SparkSession
from pyspark.sql.types import *
from pyspark.sql.functions import *
from pyspark import SparkContext, SparkConf
```

In [0]:

```
spark = SparkSession.builder.getOrCreate()
sc = spark.sparkContext
```

In [0]:

```
filename = '/FileStore/tables/kindle_reduced_clean-3.csv'
df = spark.read.csv(filename,  inferSchema=True, header = True)
```

In [0]:

```
df.select("overall","summary","reviewText").show(5)
```

```
+-------+--------------------+--------------------+
overall|             summary|          reviewText|
+-------+--------------------+--------------------+
      5|  A Very Sexy Cruise|ARC provided by a...|
      5|A Changing Gears ...|Wild Ride by Nanc...|
      5|We don't take kin...|Well thought out ...|
      3|Mediocre Science ...|Being autistic, I...|
      3| I'm losing interest|This is book four...|
+-------+--------------------+--------------------+
only showing top 5 rows
```

In [0]:

```
df.select([count(when(col(c).isNull(), c)).alias(c) for c in df.columns]).show()
```

```
+-----+----+-------+-------+----------+----------+----------+------------+-------+--------------+-
------------+----------+--------------+
index|asin|helpful|overall|reviewText|reviewTime|reviewerID|reviewerName|summary|unixReviewTime|He
pfulRecords|HasHelpful|weightedRating|
+-----+----+-------+-------+----------+----------+----------+------------+-------+--------------+-
------------+----------+--------------+
    0|   0|      0|      0|         1|         0|         0|          24|      0|             0|
0|         0|             0|
+-----+----+-------+-------+----------+----------+----------+------------+-------+--------------+-
------------+----------+--------------+
```

In [0]:

```
df = df.dropna(how='any')
```

In [0]:

```
df=df.drop("index","reviewerName","unixReviewTime","helpful","HasHelpful")
```

In [0]:

```
df = df.withColumn('reviewText', translate('reviewText', '.', ''))
```

```
df = df.withColumn('reviewText', translate('reviewText', ',', ''))
df = df.withColumn('reviewText', translate('reviewText', '$', ''))
```

In [0]:

```
from pyspark.ml.feature import Tokenizer, StopWordsRemover

#tokenize text (make words into an array)
tokenizer = Tokenizer(inputCol='reviewText', outputCol='reviewText_token')
df_token = tokenizer.transform(df).select('*')

#remove basic words
remover = StopWordsRemover(inputCol='reviewText_token', outputCol='reviewText_clean')
df_stop=remover.transform(df_token).select('*')
```

In [0]:

```
#tokenize summaries (make words into an array)
tokenizer = Tokenizer(inputCol='summary', outputCol='summary_token')
df_token = tokenizer.transform(df_stop).select('*')

#remove basic words
remover = StopWordsRemover(inputCol='summary_token', outputCol='summary_clean')
df_stop=remover.transform(df_token).select('*')
```

In [0]:

```
df_stop=df_stop.drop("reviewText", "summary","reviewText_token", "summary_token")
df_stop.show(5)
```

```
+----------+-------+-----------+--------------+-------------+--------------+--------------------+--------------------+
      asin|overall| reviewTime|    reviewerID|HelpfulRecords|weightedRating|    reviewText_clean|summary_clean|
+----------+-------+-----------+--------------+-------------+--------------+--------------------+--------------------+
B00J4S6YWC|      5|06 21, 2014| AUSBN91MCI3WM|          0.0|           5.0|[arc, provided, a...|[sexy, cruise]|
B00HCZUBH8|      5| 03 3, 2014|A141H51I3H4B1S|          0.5|           5.0|[wild, ride, nanc...|[changing, gears,...|
B006RZNR3Y|      5|07 10, 2014| AP8TKDM76TROZ|          0.0|           4.0|[well, thought, s...|[take, kindly, no!]|
B006RZNR3Y|      3| 02 1, 2014|A22GGHISKRVAOX|          0.0|           4.0|[autistic, freque...|[mediocre, scienc...|
B00J47H8H8|      3|03 21, 2014|A19DWIC1T7127Y|         0.75|           3.0|[book, four, five...|[losing, interest]|
+----------+-------+-----------+--------------+-------------+--------------+--------------------+--------------------+
only showing top 5 rows
```

In [0]:

```
display(df_stop.select("reviewText_clean"))
```

reviewText_clean
List(arc, provided, author, exchange, honest, reviewthis, first, time, read, book, miranda, p, charles, lastthis, book, -, broken, hearts, twisted, stories, lies, scared, confused, lovers, zach, rebecca, met, hired, private, nurse, grandmother, surgery, dated, four, months, rebecca, said, three, little, words, zach, say, back, rebecca, breaks, chance, tomonths, later, hired, join, grandmother, month, long, cruise, birthday, nurse, grandmother, also, friends, grandsons, joining, birthday, celebration, , cruise, around, rebecca, zach, work, fears, misunderstandings, find, love, again?a, sexy, quick, read, able, put, down!!!)
List(wild, ride, nancy, warrenchanging, gears, seriesduncan, forbes, professor, sabbatical, writes, searches, lost, stolen, art, following, lead, long, lost, van, gogh, leads, small, town, swiftcurrent, oregonwith, sexiest, librarian, ever, seenalexandra, forrest, agenda, grandfather, passed, away, plans, finishing, details, writing, memoirs, packing, home, sell, move, big, city, complete, life, plan, one, definitely, include, sexy, stranger, librarygillian, forrest, munn, messed, life, big, time, teen, small, towns, forget,

reviewText_clean
husband, left, alone, officer, tom, perkins, seems, really, believe, changeda, dead, body, library, quiet, town, changes, everything, things, like, happen, steamy, romance, murder, mystery, two, happenings, going, another, page, turner, story**strong, sexual, content, language)
List(well, thought, story, many, things, going, time, alien, race, jumps, earth, orbit, destroys, major, earth, cities, tells, us, stop, technologies, fun, begins!, helpful, alien, side, love, story, twist!, good, read!)
List(autistic, frequent, reading, difficulties, especially, third-person, stories, though, read, lacuna, twice, still, appreciate,

In [0]:

```
df_stop.printSchema()
```

```
root
-- asin: string (nullable = true)
-- overall: integer (nullable = true)
-- reviewTime: string (nullable = true)
-- reviewerID: string (nullable = true)
-- HelpfulRecords: double (nullable = true)
-- weightedRating: double (nullable = true)
-- reviewText_clean: array (nullable = true)
    |-- element: string (containsNull = true)
-- summary_clean: array (nullable = true)
    |-- element: string (containsNull = true)
```

In [0]:

```
df_stop.show(5)
```

```
+----------+-------+-----------+--------------+--------------+--------------+--------------------
------------------+
      asin|overall| reviewTime|    reviewerID|HelpfulRecords|weightedRating|    reviewText_clean|
summary_clean|
+----------+-------+-----------+--------------+--------------+--------------+--------------------
------------------+
B00J4S6YWC|      5|06 21, 2014| AUSBN91MCI3WM|           0.0|           5.0|[arc, provided, a...|
[sexy, cruise]|
B00HCZUBH8|      5| 03 3, 2014|A141H51I3H4B1S|           0.5|           5.0|[wild, ride, nanc...|
[changing, gears,...|
B006RZNR3Y|      5|07 10, 2014| AP8TKDM76TROZ|           0.0|           4.0|[well, thought, s...|
[take, kindly, no!]|
B006RZNR3Y|      3| 02 1, 2014|A22GGHISKRVAOX|           0.0|           4.0|[autistic, freque...|
[mediocre, scienc...|
B00J47H8H8|      3|03 21, 2014|A19DWIC1T7127Y|          0.75|           3.0|[book, four, five...|
[losing, interest]|
+----------+-------+-----------+--------------+--------------+--------------+--------------------
------------------+
only showing top 5 rows
```

In [0]:

```
#Exploratory Data Analysis
```

In [0]:

```
df_stop.describe().show()
```

```
+-------+----------+-----------------+----------+-------------+------------------+-----------------
-+
summary|      asin|          overall|reviewTime|   reviewerID|    HelpfulRecords|    weightedRati
ng|
+-------+----------+-----------------+----------+-------------+------------------+-----------------
-+
  count|      4880|             4880|      4880|         4880|              4880|              488
```

```
0|
   mean|      null|4.340573770491804|      null|         null|0.3715991527158007|  4.340971085027
69|
 stddev|      null|0.973934363172232|      null|
null|0.4611430329911328|0.9374090879340996|
    min|B000SRGF2W|                1|01 1, 2011| A0JVI0NYIOT2|               0.0|                1
.0|
    max|B00LYPZIXO|                5|12 9, 2013|AZZFLSL2LE4FX|               1.0| 5.0000000000000
01|
+-------+----------+-----------------+----------+-------------+------------------+-----------------
-+
```

In [0]:

```
#Start of Pipelines
```

In [0]:

```
from pyspark import HiveContext
hiveContext = HiveContext(sc)

#df.show(truncate = False)
# Get term frequency vector through HashingTF
from pyspark.ml.feature import HashingTF
ht = HashingTF(inputCol="reviewText_clean", outputCol="review_features")
result = ht.transform(df_stop)
ht1 = HashingTF(inputCol="summary_clean", outputCol="summary_features")
result = ht1.transform(result)
result.show(2)
```

```
+----------+-------+-----------+--------------+--------------+--------------+--------------------
------------------+--------------------+--------------------+
      asin|overall| reviewTime|    reviewerID|HelpfulRecords|weightedRating|    reviewText_clean|
summary_clean|     review_features|    summary_features|
+----------+-------+-----------+--------------+--------------+--------------+--------------------
------------------+--------------------+--------------------+
B00J4S6YWC|      5|06 21, 2014| AUSBN91MCI3WM|           0.0|           5.0|[arc, provided, a...|
[sexy, cruise]|(262144,[1546,119...|(262144,[16757,84...|
B00HCZUBH8|      5| 03 3, 2014|A141H51I3H4B1S|           0.5|           5.0|[wild, ride, nanc...|
[changing, gears,...|(262144,[6346,687...|(262144,[50415,13...|
+----------+-------+-----------+--------------+--------------+--------------+--------------------
------------------+--------------------+--------------------+
only showing top 2 rows
```

In [0]:

```
result=result.drop("reviewText_clean","summary_clean")
```

In [0]:

```
type(result)
```

Out[20]: pyspark.sql.dataframe.DataFrame

In [0]:

```
df_sp = result.withColumn('overall', when(result.overall >= 2.5,1).otherwise(0))
#df_sp = df_sp.withColumn('HasHelpful', when(df_sp.HasHelpful == True,1).otherwise(0))
```

In [0]:

```
df_sp.show(3)
```

```
+----------+-------+----------+--------------+-------------+-------------+--------------------+--------------------+
      asin|overall| reviewTime|    reviewerID|HelpfulRecords|weightedRating|     review_features|    summary_features|
+----------+-------+----------+--------------+-------------+-------------+--------------------+--------------------+
B00J4S6YWC|      1|06 21, 2014| AUSBN91MCI3WM|          0.0|          5.0|(262144,[1546,119...|(262144,[16757,84...|
B00HCZUBH8|      1| 03 3, 2014|A141H51I3H4B1S|          0.5|          5.0|(262144,[6346,687...|(262144,[50415,13...|
B006RZNR3Y|      1|07 10, 2014| AP8TKDM76TROZ|          0.0|          4.0|(262144,[2325,230...|(262144,[51852,55...|
+----------+-------+----------+--------------+-------------+-------------+--------------------+--------------------+
only showing top 3 rows
```

In [0]:

```
from pyspark.ml.feature import StringIndexer
from pyspark.ml.feature import StandardScaler
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import OneHotEncoder
from pyspark.mllib.linalg import Vectors
```

In [0]:

```
df_sp.printSchema()
```

```
root
-- asin: string (nullable = true)
-- overall: integer (nullable = false)
-- reviewTime: string (nullable = true)
-- reviewerID: string (nullable = true)
-- HelpfulRecords: double (nullable = true)
-- weightedRating: double (nullable = true)
-- review_features: vector (nullable = true)
-- summary_features: vector (nullable = true)
```

In [0]:

```
# label encode

asin_indexer = StringIndexer(inputCol = 'asin', outputCol='asin_num').setHandleInvalid("
skip").fit(df_sp)
df_sp = asin_indexer.transform(df_sp)

reviewTime_indexer = StringIndexer(inputCol = 'reviewTime', outputCol='reviewTime_num').s
etHandleInvalid("skip").fit(df_sp)
df_sp = reviewTime_indexer.transform(df_sp)

reviewerID_indexer = StringIndexer(inputCol = 'reviewerID', outputCol='reviewerID_num').s
etHandleInvalid("skip").fit(df_sp)
df_sp = reviewerID_indexer.transform(df_sp)
```

In [0]:

```
asin_onehoter = OneHotEncoder(inputCol='asin_num', outputCol='asin_vector').fit(df_sp)
df_sp = asin_onehoter.transform(df_sp)

reviewTime_onehoter = OneHotEncoder(inputCol='reviewTime_num', outputCol='reviewTime_vect
or').fit(df_sp)
df_sp = reviewTime_onehoter.transform(df_sp)
```

```
reviewerID_onehoter = OneHotEncoder(inputCol='reviewerID_num', outputCol='reviewerID_vect
or').fit(df_sp)
df_sp = reviewerID_onehoter.transform(df_sp)
```

In [0]:

```
display(df_sp)
```

asin	overall	reviewTime	reviewerID	HelpfulRecords	weightedRating	review_features	sum
						Map(vectorType -> sparse, length -> 262144, indices -> List(1546, 11941, 16757, 25764, 34343, 38640, 39143, 41931, 42882, 45155, 50793, 53570, 60345, 68044, 71961, 75181, 75836, 77751, 81103, 84028, 84696, 84933, 90859,	

In [0]:

```
df_assem = VectorAssembler(inputCols=['review_features','summary_features','weightedRatin
g','asin_vector','reviewTime_vector',
                                      'reviewerID_vector'],
                                      outputCol='features')
df_assem = df_assem.transform(df_sp)
```

In [0]:

```
#Train Test Split
seed = 314
train_test = [0.8, 0.2]


data_set = df_assem.select(['features','overall'])
train_df, test_df = data_set.randomSplit(train_test, seed)
```

In [0]:

```
#Linear Regression
```

In [0]:

```
from pyspark.ml.regression import LinearRegression
lr = LinearRegression(featuresCol = 'features', labelCol='overall', maxIter=10, regParam
=0.3, elasticNetParam=0.8)
lr_model = lr.fit(train_df)
```

In [0]:

```
print("Coefficients: " + str(lr_model.coefficients))
print("Intercept: " + str(lr_model.intercept))
```

```
Coefficients: (534039,[],[])
Intercept: 0.9383861820056716
```

In [0]:

```
trainingSummary = lr_model.summary
print("RMSE: %f" % trainingSummary.rootMeanSquaredError)
```

```
RMSE: 0.240453
```

In [0]:

```
train_df.describe().show()
```

```
+-------+-------------------+
summary|            overall|
+-------+-------------------+
  count|               3879|
   mean| 0.9383861820056716|
 stddev|0.24048381347486347|
    min|                  0|
    max|                  1|
+-------+-------------------+
```

In [0]:

```
lr_predictions = lr_model.transform(test_df)
lr_predictions.select("prediction","overall","features").show(5)
from pyspark.ml.evaluation import RegressionEvaluator
lr_evaluator = RegressionEvaluator(predictionCol="prediction", \
                  labelCol="overall",metricName="r2")
```

```
+------------------+-------+--------------------+
        prediction|overall|            features|
+------------------+-------+--------------------+
0.9383861820056716|      1|(534039,[19,2437,...|
0.9383861820056716|      1|(534039,[65,3280,...|
0.9383861820056716|      1|(534039,[90,2015,...|
0.9383861820056716|      1|(534039,[135,3924...|
0.9383861820056716|      1|(534039,[143,2701...|
+------------------+-------+--------------------+
only showing top 5 rows
```

In [0]:

```
test_result = lr_model.evaluate(test_df)
print("Root Mean Squared Error (RMSE) on test data = %g" % test_result.rootMeanSquaredErr
or)
```

```
Root Mean Squared Error (RMSE) on test data = 0.255165
```

In [0]:

```
from pyspark.ml.classification import LogisticRegression

# Train Logistic Regression Model
log_reg = LogisticRegression(labelCol = 'overall').fit(train_df)

train_pred = log_reg.evaluate(train_df).predictions

train_pred.filter(train_pred['overall'] == 1).filter(train_pred['prediction'] == 1).sele
ct(['overall', 'prediction', 'probability']).show(10, False)
```

```
+-------+----------+-----------------------------------------+
overall|prediction|probability                              |
+-------+----------+-----------------------------------------+
1      |1.0       |[2.1317615376959238E-10,0.9999999997868239]|
1      |1.0       |[6.337259584813845E-10,0.999999999366274]  |
1      |1.0       |[2.3867270923490733E-11,0.9999999999761326]|
1      |1.0       |[3.382793656631829E-12,0.9999999999966172] |
1      |1.0       |[2.7449081140086084E-17,1.0]               |
1      |1.0       |[1.677982205171266E-15,0.9999999999999982] |
1      |1.0       |[1.6542055000529485E-9,0.9999999983457946] |
1      |1.0       |[2.0426642044809184E-10,0.9999999979573361]|
```

```
1      |1.0       |[1.6582182443850235E-9,0.9999999983417818] |
1      |1.0       |[1.1370137405333302E-11,0.9999999999886298]|
+-------+----------+-------------------------------------------+
only showing top 10 rows
```

In [0]:

```
# Evaluate on testdata

test_result = log_reg.evaluate(test_df).predictions
test_result.show(3)
```

```
+--------------------+-------+--------------------+--------------------+----------+
            features|overall|       rawPrediction|         probability|prediction|
+--------------------+-------+--------------------+--------------------+----------+
(534040,[19,2437,...|      1|[-20.599055811585...|[1.13225364672783...|       1.0|
(534040,[65,3280,...|      1|[-23.369621314796...|[7.09091727506984...|       1.0|
(534040,[90,2015,...|      1|[-19.989701241476...|[2.08249062528958...|       1.0|
+--------------------+-------+--------------------+--------------------+----------+
only showing top 3 rows
```

In [0]:

```
# Accuracy computation

tp = test_result[(test_result.overall == 1) & (test_result.prediction == 1)].count()
tn = test_result[(test_result.overall == 0) & (test_result.prediction == 1)].count()
fp = test_result[(test_result.overall == 0) & (test_result.prediction == 1)].count()
fn = test_result[(test_result.overall == 1) & (test_result.prediction == 0)].count()

print('test accuracy is : %f'%((tp+tn)/(tp+tn+fp+fn)))
```

```
test accuracy is : 0.937912
```

In [0]:

```
# Recall and Precision

print('test recall is : %f'%(tp/(tp+fn)))
print('test precision is : %f'%(tp/(tp+fp)))
```

```
test recall is : 1.000000
test precision is : 0.933801
```

In [0]:

```
# F1 score

recall = tp/(tp+fn)
precision = tp/(tp+fp)

F1 =  2 * (precision*recall) / (precision + recall)
print('F1 score: %0.3f' % F1)
```

```
F1 score: 0.966
```

In [0]:

```
#NaiveBayes
```

In [0]:

```
from pyspark.ml.classification import NaiveBayes
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

train_df = train_df.withColumnRenamed('overall','label')
test_df = test_df.withColumnRenamed('overall','label')
```

```
# create the trainer and set its parameters
nb = NaiveBayes(smoothing=1.0, modelType="multinomial")

# train the model
model = nb.fit(train_df)

# select example rows to display.
predictions = model.transform(test_df)
predictions.show()

# compute accuracy on the test set
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction
",
                                              metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test set accuracy = " + str(accuracy))
```

```
+--------------------+-----+--------------------+--------------------+----------+
            features|label|       rawPrediction|         probability|prediction|
+--------------------+-----+--------------------+--------------------+----------+
(534040,[19,2437,...|    1|[-1333.2960363899...|[4.37136657361991...|       1.0|
(534040,[65,3280,...|    1|[-1072.5032970387...|[6.34692467581792...|       1.0|
(534040,[90,2015,...|    1|[-2189.7649668781...|[6.88453580775355...|       1.0|
(534040,[135,3924...|    1|[-410.39737084138...|[9.20586134263801...|       1.0|
(534040,[143,2701...|    1|[-1535.4022614037...|[2.71754092205643...|       1.0|
(534040,[156,2306...|    1|[-369.45287642239...|[1.33652276552001...|       1.0|
(534040,[216,2488...|    1|[-556.68601081198...|[2.41946672329676...|       1.0|
(534040,[233,3564...|    1|[-1407.3915693416...|[3.96609601713866...|       1.0|
(534040,[288,4914...|    1|[-798.28518421547...|[1.39362199422467...|       1.0|
(534040,[288,6498...|    1|[-850.62944383437...|[3.33887687901917...|       1.0|
(534040,[324,4614...|    1|[-969.08757503740...|[4.20133823611772...|       1.0|
(534040,[329,619,...|    1|[-4005.5698578446...|[1.76864439776849...|       1.0|
(534040,[329,1889...|    1|[-1561.1607244568...|[2.84351967262817...|       1.0|
(534040,[329,2437...|    1|[-1025.7593313519...|[7.06232143838649...|       1.0|
(534040,[329,1463...|    1|[-831.05851927255...|[6.79050399014597...|       1.0|
(534040,[332,1074...|    1|[-2622.8118759576...|[8.61773856317226...|       1.0|
(534040,[332,1889...|    1|[-1982.9592306513...|[1.24192163648331...|       1.0|
(534040,[378,411,...|    1|[-3551.4606100962...|[1.83558311883560...|       1.0|
(534040,[379,5755...|    1|[-1226.6388625845...|[6.67078353116071...|       1.0|
(534040,[404,3524...|    1|[-1225.5025534758...|[2.58727861801961...|       1.0|
+--------------------+-----+--------------------+--------------------+----------+
only showing top 20 rows

Test set accuracy = 0.9300699300699301
```

In [0]:

```
# Accuracy computation

tp = predictions[(predictions.label == 1) & (predictions.prediction == 1)].count()
tn = predictions[(predictions.label == 0) & (predictions.prediction == 1)].count()
fp = predictions[(predictions.label == 0) & (predictions.prediction == 1)].count()
fn = predictions[(predictions.label == 1) & (predictions.prediction == 0)].count()

recall = tp/(tp+fn)
precision = tp/(tp+fp)

# Recall and Precision

print('test recall is : %f'% recall)
print('test precision is : %f'% precision)
```

```
test recall is : 1.000000
test precision is : 0.930070
```

In [0]:

```
# F1 score
```

```
# F1 score

F1 =  2 * (precision*recall) / (precision + recall)
print('F1 score: %0.3f' % F1)
```

```
F1 score: 0.964
```

In [0]:

```
#Random Forest
```

In [0]:

```
#train RF model
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

rf = RandomForestClassifier(labelCol = 'label', featuresCol = "features", numTrees = 20,
seed = 314)
```

In [0]:

```
model = rf.fit(train_df)
```

In [0]:

```
pred = model.transform(test_df)
```

In [0]:

```
evalRF = MulticlassClassificationEvaluator(labelCol = 'label', predictionCol = "predictio
n", metricName = "accuracy")
```

In [0]:

```
acc = evalRF.evaluate(pred)
print("Test set accuracy = " + str(acc))
```

```
Test set accuracy = 0.9300699300699301
```

In [0]:

```
# Accuracy computation

tp = pred[(pred.label == 1) & (pred.prediction == 1)].count()
tn = pred[(pred.label == 0) & (pred.prediction == 1)].count()
fp = pred[(pred.label == 0) & (pred.prediction == 1)].count()
fn = pred[(pred.label == 1) & (pred.prediction == 0)].count()

recall = tp/(tp+fn)
precision = tp/(tp+fp)

# Recall and Precision

print('test recall is : %f'% recall)
print('test precision is : %f'% precision)
```

```
test recall is : 1.000000
test precision is : 0.930070
```

In [0]:

```
# F1 score

F1 =  2 * (precision*recall) / (precision + recall)
print('F1 score: %0.3f' % F1)
```

```
F1 score: 0.964
```

In [0]:

```
```