In [1]:
```
#Chinmay Bake Machine Learning Assignment 5B
```

Clear Statement of the problem to solve

Our objective is to predict origin of wines, either US or Non-US, based on the Description from the given historical data. So the bottomline is that we input the description given in the dataset using various NLP processes, so that they are transformed into a numerical vector or would essentially be our feature variable. Our label, is categorical and has two categories to be precise. Finally, our Machine Learning Model would classify the category of the label variable based on the patterns it has learnt from the historical data.

In [65]:
```
from pyspark.sql import SparkSession
from pyspark.ml.feature import StopWordsRemover
from pyspark.ml.feature import Tokenizer
from pyspark.sql.functions import regexp_replace, trim, col, lower
from pyspark.sql.functions import rand
from pyspark.sql.functions import length
from pyspark.ml.feature import StopWordsRemover
from pyspark.sql.functions import rand
from pyspark.sql.functions import length
```

Statement/Comment on the data collection (in terms of what is needed) and scope

We must be having written description against each category of origin of wines. The description for a specific category, for instance US wines, would have certain patterns in its description; perhaps recurring words or phrases. We expect our machine learning model to capture this underlying trend within the description and make predictions based on that. We also need the categories to our target variable to be labelled with certain numeric values.

In [62]:
```
spark=SparkSession.builder.appName('ML_NLP').getOrCreate()
main_df=spark.read.csv(r'D:\ShortWineReviews.csv', inferSchema=True,header=Tru
e, sep=',')
main_df.show()
main_df.printSchema()
```

```
+------+--------------------+
|Origin|         Description|
+------+--------------------+
|    US|This tremendous 1...|
|    US|Mac Watson honors...|
|    US|This spent 20 mon...|
|    US|This re-named vin...|
|    US|The producer sour...|
|    US|From 18-year-old ...|
|    US|A standout even i...|
|    US|With its sophisti...|
|    US|First made in 200...|
|    US|This blockbuster,...|
|    US|This fresh and li...|
|    US|Heitz has made th...|
|    US|The apogee of thi...|
|    US|San Jose-based pr...|
|    US|Bergström has mad...|
|    US|Focused and dense...|
|    US|Cranberry, baked ...|
|    US|This standout Roc...|
|    US|Steely and perfum...|
|    US|The aromas entice...|
+------+--------------------+
only showing top 20 rows

root
 |-- Origin: string (nullable = true)
 |-- Description: string (nullable = true)
```

Preliminary decision about the ML algorithm you will use. Justify your decision to use this particular algorithm

Our objective is to predict the categories based on given descriptions. Hence, we need a Classification algorithm. Although, the data we have in our target variable is not labelled, we would perform certain tranformations (StringIndexing) so that each category could get a numerical value. For classification, we could either use Logistic Regression or Random Forests. We could try using Random Forests, as the algorithm is more tolerant towards missing values and presence of outliers within the dataset. As we have not weighed a lot upon outlier analysis and missing value treatment in this specific case, we would use Random Forests for classifying our categories.

In [64]:
```
# Perform data prep and clean-up.

def removePunctuation(column):
    return trim(lower(regexp_replace(column, '([^\s\w_]|_)+', '')))

main_df= main_df.withColumn('Description_NoPunct', removePunctuation(col('Desc
ription')))

tokenization=Tokenizer(inputCol='Description_NoPunct',outputCol='Tokens')
tokenized_main_df=tokenization.transform(main_df)

stopword_removal=StopWordsRemover(inputCol='Tokens',outputCol='NoStWords_Token
s')
NoStWords_df=stopword_removal.transform(tokenized_main_df)
```

Statement/comments on data preparation and clean up. What do you need to do to get data ready for the ML algorithm of your choice.

We perform three main steps on the text data before processing it into numeric format. First we define a set of characters and strip them of the data as a part of removing punctuations. Later, we seperate each word of a sentence individually to form tokenized data. Finally, words which do not have much significance when text analysing are pulled out of the dataset during the stopwords removal process.

In [67]:
```
# Conduct Explanatory data analysis (EDA). Comment on data distribution.

main_df.count()
```

Out[67]:
```
4000
```

In [69]:
```
main_df=main_df.filter(((main_df.Origin =='US') | (main_df.Origin =='Non-US'
)))
main_df.count()
```

Out[69]:
```
4000
```

In [70]:
```
main_df=main_df.withColumn('length',length(main_df['Description_NoPunct']))
main_df.orderBy(rand()).show(5,True)
main_df.groupBy('Origin').agg({'Length':'mean'}).show()
```

```
+------+--------------------+--------------------+------+
|Origin|         Description| Description_NoPunct|length|
+------+--------------------+--------------------+------+
|Non-US|Aromas that recal...|aromas that recal...|   305|
|Non-US|Fruity and attrac...|fruity and attrac...|   131|
|Non-US|A full-bodied win...|a fullbodied wine...|   216|
|    US|Dark red cherry a...|dark red cherry a...|   297|
|    US|Only two acres of...|only two acres of...|   400|
+------+--------------------+--------------------+------+
only showing top 5 rows

+------+-----------+
|Origin|avg(Length)|
+------+-----------+
|Non-US|   245.1995|
|    US|   246.5665|
+------+-----------+
```

Conduct Explanatory data analysis (EDA). Comment on data distribution.

We first ensure that thhere is consitency in the categorical target varialbe across our dataset. Hence we recount the number of records after filtering the dataset with known categories. We can observe that there isn't much of a difference in the mean length of description for both our categories. This possibly indicates that the machine learning won't have excessive learning scope from a specif category as our data is well distrubuted in terms on the lenght of reviews.

In [72]:
```
# Data Transformation

NoStWords_df.printSchema()

from pyspark.ml.feature import StringIndexer
indexer = StringIndexer(inputCol="Origin", outputCol="OriginIndexLabel")
indexed_df = indexer.fit(NoStWords_df).transform(NoStWords_df)
indexed_df.select(['NoStWords_Tokens','OriginIndexLabel']).show(10)
```

```
root
 |-- Origin: string (nullable = true)
 |-- Description: string (nullable = true)
 |-- Description_NoPunct: string (nullable = true)
 |-- Tokens: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- NoStWords_Tokens: array (nullable = true)
 |    |-- element: string (containsNull = true)

+--------------------+----------------+
|    NoStWords_Tokens|OriginIndexLabel|
+--------------------+----------------+
|[tremendous, 100,...|             1.0|
|[mac, watson, hon...|             1.0|
|[spent, 20, month...|             1.0|
|[renamed, vineyar...|             1.0|
|[producer, source...|             1.0|
|[18yearold, vines...|             1.0|
|[standout, even, ...|             1.0|
|[sophisticated, m...|             1.0|
|[first, made, 200...|             1.0|
|[blockbuster, pow...|             1.0|
+--------------------+----------------+
only showing top 10 rows
```

Comment on required data transformation needed to get the data ready for input to the ML algorithm of your choice

As illustrated in our dataset's schema, the target variable is categorical. Hence, we map categrical values of that variable against certain numeric values using the StringIndexer function. The resultant column has a numeric data type.

In [75]:
```
from pyspark.ml.feature import HashingTF,IDF
hashing_vector=HashingTF(inputCol='NoStWords_Tokens',outputCol='Features')
hashing_df=hashing_vector.transform(indexed_df)

tf_idf_vec=IDF(inputCol='Features',outputCol='IDF_Features')
tf_idf_df=tf_idf_vec.fit(hashing_df).transform(hashing_df)

model_df=tf_idf_df.select(['OriginIndexLabel','IDF_Features'])

from pyspark.ml.feature import VectorAssembler
idf_df_assembler = VectorAssembler(inputCols=['IDF_Features'],outputCol='IDF_F
eatures_Vec')
model_df= idf_df_assembler.transform(model_df)

new_model_df=model_df.select(col("IDF_Features_Vec").alias("features"),("Origi
nIndexLabel"))
new_model_df.show()
```

```
+--------------------+----------------+
|            features|OriginIndexLabel|
+--------------------+----------------+
|(262144,[5358,204...|             1.0|
|(262144,[40266,43...|             1.0|
|(262144,[15519,23...|             1.0|
|(262144,[3189,319...|             1.0|
|(262144,[15664,21...|             1.0|
|(262144,[32653,35...|             1.0|
|(262144,[4176,235...|             1.0|
|(262144,[5460,157...|             1.0|
|(262144,[13114,31...|             1.0|
|(262144,[35120,42...|             1.0|
|(262144,[15786,21...|             1.0|
|(262144,[17200,26...|             1.0|
|(262144,[571,1566...|             1.0|
|(262144,[932,3007...|             1.0|
|(262144,[4176,546...|             1.0|
|(262144,[20,614,2...|             1.0|
|(262144,[551,2851...|             1.0|
|(262144,[5765,606...|             1.0|
|(262144,[14114,14...|             1.0|
|(262144,[47075,61...|             1.0|
+--------------------+----------------+
only showing top 20 rows
```

Complete all needed data transformation. Use TF-IDF method of transforming token to their respective numeric values.

We first convert the texts into numeric equivalents using the TF-IDF method and then use VectorAssembler for vectorizing the TF-IDF features

In [76]:
```
# Apply the ML algorithm of your choice. Use an 8-/20 split for creating train
ing and test dataset.

training_df,test_df=new_model_df.randomSplit([0.80,0.20])
from pyspark.ml.classification import RandomForestClassifier
rf_classifier=RandomForestClassifier(labelCol='OriginIndexLabel',numTrees=50).
fit(training_df)
```

In [78]:
```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.evaluation import BinaryClassificationEvaluator

training_predictions=rf_classifier.transform(training_df)

rf_accuracy=MulticlassClassificationEvaluator(labelCol='OriginIndexLabel',metr
icName='accuracy').evaluate(training_predictions)
print('Training Accuracy: ',rf_accuracy)

rf_precision=MulticlassClassificationEvaluator(labelCol='OriginIndexLabel',met
ricName='weightedPrecision').evaluate(training_predictions)
print('Training Precision: ', rf_precision)

rf_auc=BinaryClassificationEvaluator(labelCol='OriginIndexLabel').evaluate(tra
ining_predictions)
print('Area Under Curve Training: ', rf_auc)
```

```
Training Accuracy:  0.8171193935565382
Training Precision:  0.8190462721719429
Area Under Curve Training:  0.906288709574427
```

Train the model using the training dataset. Comment on the overall model fit using evaluation metrics of your chosen ML algorithms. Comment on your findings and draw conclusions about the trained model's accuracy and worthiness.

Our model yeilds an accuracy of 81% which in turn is satisfactory. We observe a high AUC value for the training data which might indicate that majority of True Positives are classified correctly.

In [80]:
```
rf_predictions=rf_classifier.transform(test_df)


test_rf_accuracy=MulticlassClassificationEvaluator(labelCol='OriginIndexLabel'
,metricName='accuracy').evaluate(rf_predictions)
print('Testing Accuracy: ',test_rf_accuracy)

test_rf_precision=MulticlassClassificationEvaluator(labelCol='OriginIndexLabe
l',metricName='weightedPrecision').evaluate(rf_predictions)
print('Testing Precision: ', test_rf_precision)

test_rf_auc=BinaryClassificationEvaluator(labelCol='OriginIndexLabel').evaluat
e(rf_predictions)
print('Area Under Curve Testing: ', test_rf_auc)
```

```
Testing Accuracy:  0.7817745803357314
Testing Precision:  0.7831011480520992
Area Under Curve Testing:  0.8847100782328577
```

11. Test/Evaluate your trained model with test dataset. Again, evaluate the model's performance using the evaluation metrics available for your ML algorithm. Again, comment on your findings and draw conclusions about the model's accuracy and worthiness in terms solving the problem you were trying to solve.

We observe not much of a difference between the training and testing metrics. This clearly indicates that the model has not overfitted the curve and is not biased. The underlying pattern in the description with NLP pipelines which we have implemented could predict categories of the target variable with fair accuracy and precision.