In [1]:
```
import psycopg2
import csv

# Connect to the PostgreSQL database
conn = psycopg2.connect(
    database="beer",
    user="postgre",
    password="admin",
    host="localhost",
    port="5432"
)

# Create a cursor object to execute SQL queries
cur = conn.cursor()
```

In [2]:
```
# Create  table
cur.execute("""
CREATE TABLE BeersAbv ( id integer,
                        ibu integer,
                        style varchar(100),
                        brewery_id integer,
                        abv decimal DEFAULT NULL
                       );
            """)

cur.execute("""
CREATE TABLE BeerOun ( id integer,
                       ibu integer,
                       name varchar(255),
                       brewery_id integer,
                       ounces decimal
                      );
            """)

cur.execute("""
CREATE TABLE Breweries ( id integer,
                         name varchar(255),
                         city varchar(100),
                         state varchar(10)
                       );
            """)
```

In [4]:
```
# Import data into table
with open('beers_abv.csv', 'r') as file:
    data = csv.reader(file)
    next(data)
    for r in data:
        r = [None if x == '' else x for x in r]
        cur.execute("INSERT INTO beersabv (id, ibu, style, brewery_id,


with open('beers_oun.csv', 'r') as file:
    data = csv.reader(file)
    next(data)
    for r in data:
        r = [None if x == '' else x for x in r]
        cur.execute("INSERT INTO BeerOun (id, ibu, name, brewery_id, ou


with open('breweries.csv', 'r') as file:
    data = csv.reader(file)
    next(data)
    for r in data:
        r = [None if x == '' else x for x in r]
        cur.execute("INSERT INTO Breweries (id, name, city, state) VALU
```

In [2]:
```
import pandas as pd
import warnings
warnings.filterwarnings("ignore", category=UserWarning, module="pandas"

# use pandas read_sql() function to fetch the data from PostgreSQL and
q = "SELECT * FROM BeersAbv"
dataframe = pd.read_sql(q, con=conn)
print(dataframe.head())
```

```
     id  ibu                          style  brewery_id    abv
0  1436  NaN            American Pale Lager         408  0.050
1  2265  NaN        American Pale Ale (APA)         177  0.066
2  2264  NaN                    American IPA         177  0.071
3  2263  NaN  American Double / Imperial IPA         177  0.090
4  2262  NaN                    American IPA         177  0.075
```

In [3]:
```
#Commit the Changes
conn.commit()

# Close the cursor and connection
cur.close()
conn.close()
```

In [4]:
```
#Select only the required Columns
dataframe = dataframe.drop(columns=['id', 'brewery_id'])
dataframe
```

Out[4]:

	ibu	style	abv
0	NaN	American Pale Lager	0.050
1	NaN	American Pale Ale (APA)	0.066
2	NaN	American IPA	0.071
3	NaN	American Double / Imperial IPA	0.090
4	NaN	American IPA	0.075
...	...	...	...
2405	45.0	Belgian IPA	0.067
2406	NaN	American Amber / Red Ale	0.052
2407	NaN	Schwarzbier	0.055
2408	40.0	American Pale Ale (APA)	0.055
2409	NaN	American Amber / Red Ale	0.052

2410 rows × 3 columns

In [5]:
```
import pyspark
from pyspark.sql import SparkSession
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.filterwarnings("ignore", category=UserWarning)


spark = SparkSession.builder \
    .appName("APP_BEER") \
    .config("spark.executor.instances", "10") \
    .config("spark.executor.memory", "8g") \
    .getOrCreate()


# convert the Pandas DataFrame to a Spark DataFrame
spark_df = spark.createDataFrame(dataframe)
spark_df.show(10)
```

```
23/04/05 16:23:50 WARN Utils: Your hostname, cis6180 resolves to a loo
pback address: 127.0.1.1; using 10.0.2.15 instead (on interface enp0s
3)
23/04/05 16:23:50 WARN Utils: Set SPARK_LOCAL_IP if you need to bind t
o another address

Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).

23/04/05 16:23:51 WARN NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where applicab
le

[Stage 0:>                                                          (0
+ 1) / 1]

+---+--------------------+-----+
|ibu|               style|  abv|
+---+--------------------+-----+
|NaN| American Pale Lager| 0.05|
|NaN|American Pale Ale...|0.066|
|NaN|        American IPA|0.071|
|NaN|American Double /...| 0.09|
|NaN|        American IPA|0.075|
|NaN|       Oatmeal Stout|0.077|
|NaN|American Pale Ale...|0.045|
|NaN|     American Porter|0.065|
|NaN|American Pale Ale...|0.055|
|NaN|American Double /...|0.086|
+---+--------------------+-----+
only showing top 10 rows
```

In [6]:
```
from pyspark.sql.functions import col

# filter the DataFrame to only include rows with style column equals to
spark_df = spark_df.filter((col('style') == 'American IPA') | (col('sty
```

In [7]:
```
spark_df = spark_df.na.drop(how='any')
spark_df.show(10)
```

```
+----+--------------------+-----+
| ibu|               style|  abv|
+----+--------------------+-----+
|60.0|American Pale Ale...|0.061|
|42.0|American Pale Ale...|0.044|
|70.0|        American IPA| 0.07|
|70.0|        American IPA| 0.07|
|70.0|        American IPA| 0.07|
|42.0|American Pale Ale...|0.044|
|65.0|        American IPA| 0.07|
|82.0|        American IPA| 0.07|
|45.0|        American IPA| 0.05|
|65.0|        American IPA|0.069|
+----+--------------------+-----+
only showing top 10 rows
```

In [8]:
```
# Import required libraries
from pyspark.ml.feature import StringIndexer, VectorAssembler
from pyspark.ml.classification import RandomForestClassifier, LinearSVC
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator


indexer = StringIndexer(inputCol="style", outputCol="label")
spark_df = indexer.fit(spark_df).transform(spark_df)

#training and testing sets
(training, test) = spark_df.randomSplit([0.6, 0.4])


#single vector column
featureCols = ["ibu", "abv"]
assembler = VectorAssembler(inputCols=featureCols, outputCol="features"

# Create algorithm object for RF and SVM
randf = RandomForestClassifier(labelCol="label", featuresCol="features"
svmachine = LinearSVC(labelCol="label", featuresCol="features")

#OneVsRest object
ovr_rf = OneVsRest(classifier=randf, labelCol="label")
ovr_svm = OneVsRest(classifier=svmachine, labelCol="label")

# Define the pipeline
pipeline_rf = Pipeline(stages=[assembler, ovr_rf])
pipeline_svm = Pipeline(stages=[assembler, ovr_svm])
```

In [9]:
```
# Define the hyperparameter grid
paramGrid_rf = ParamGridBuilder() \
    .addGrid(randf.maxDepth, [5, 10]) \
    .addGrid(randf.numTrees, [20, 50]) \
    .build()

paramGrid_svm = ParamGridBuilder() \
    .addGrid(svmachine.maxIter, [10, 20]) \
    .addGrid(svmachine.regParam, [0.1, 0.01]) \
    .build()
```

In [10]:
```
# Create a CrossValidator object for Random Forest Classifier
CrossValidator_rf = CrossValidator(estimator=pipeline_rf,
                        estimatorParamMaps=paramGrid_rf,
                        evaluator=MulticlassClassificationEvaluator(),
                        numFolds=5)

CrossValidator_svm = CrossValidator(estimator=pipeline_svm,
                         estimatorParamMaps=paramGrid_svm,
                         evaluator=MulticlassClassificationEvaluator(),
                         numFolds=5)
```

In [11]:
```
# Train the Data
model_randf = CrossValidator_rf.fit(training)
model_svm = CrossValidator_svm.fit(training)
```

```
23/04/05 16:25:55 WARN InstanceBuilder$NativeBLAS: Failed to load impl
ementation from:dev.ludovic.netlib.blas.JNIBLAS
23/04/05 16:25:55 WARN InstanceBuilder$NativeBLAS: Failed to load impl
ementation from:dev.ludovic.netlib.blas.ForeignLinkerBLAS
```

In [12]:
```
predictions_randf = model_randf.transform(test)
```

In [13]:
```
predictions_svm = model_svm.transform(test)
```

In [17]:
```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator, Bi

# evaluate accuracy
randf_accuracy = MulticlassClassificationEvaluator(labelCol="label", pr
    .setMetricName("accuracy") \
    .evaluate(predictions_randf)

svm_accuracy = MulticlassClassificationEvaluator(labelCol="label", pred
    .setMetricName("accuracy") \
    .evaluate(predictions_svm)

#Evaluate precision
randf_Precision = MulticlassClassificationEvaluator(labelCol="label", p
    .evaluate(predictions_randf)
svm_Precision = MulticlassClassificationEvaluator(labelCol="label", pre
    .evaluate(predictions_svm)

#Evaluate recall
randf_Recall= MulticlassClassificationEvaluator(labelCol="label", predi
    .evaluate(predictions_randf)
svm_Recall = MulticlassClassificationEvaluator(labelCol="label", predic
    .evaluate(predictions_svm)

#Evaluate f1-score for both classifiers
randf_f1 = MulticlassClassificationEvaluator(labelCol="label", predicti
    .evaluate(predictions_randf)
svm_f1 = MulticlassClassificationEvaluator(labelCol="label", prediction
    .evaluate(predictions_svm)

# evaluate AuROC for both classifiers
randf_auROC = BinaryClassificationEvaluator(labelCol="label", rawPredic
    .setMetricName("areaUnderROC") \
    .evaluate(predictions_randf)
svm_auROC = BinaryClassificationEvaluator(labelCol="label", rawPredicti
    .setMetricName("areaUnderROC") \
    .evaluate(predictions_svm)



print("Random Forest Accuracy: {} %".format(randf_accuracy*100))
print("Linear SVM Accuracy: {} %".format(svm_accuracy*100))
print('\n')

print("Random Forest Precision: {}".format(randf_Precision))
print("Linear SVM Precision: {}".format(svm_Precision))
print('\n')


print("Random Forest Recall: {}".format(randf_Recall))
print("Linear SVM Recall: {}".format(svm_Recall))
print('\n')

print("Random Forest F1-Score: {}".format(randf_f1))
print("Linear SVM F1-Score: {}".format(svm_f1))
print('\n')
```

```
print("Random Forest AuROC: {}".format(randf_auROC))
print("Linear SVM AuROC: {}".format(svm_auROC))
```

```
Random Forest Accuracy: 80.33707865168539 %
Linear SVM Accuracy: 80.33707865168539 %


Random Forest Precision: 0.80237778248887
Linear SVM Precision: 0.806760144860247


Random Forest Recall: 0.803370786516854
Linear SVM Recall: 0.8033707865168539


Random Forest F1-Score: 0.8028467961945439
Linear SVM F1-Score: 0.8048240857229622


Random Forest AuROC: 0.764784946236559
Linear SVM AuROC: 0.7752389486260455
```

In [18]:
```
spark.stop()
```