Truata Data Engineering Coding Challenge

Introduction

The purpose of this coding challenge is to simulate concrete data engineering scenarios arising within our usiness in two categories:

1.) Generation of usiness intelligence reports and delivery of data insights

2.) Applied machine learning

Guidelines

- Your code must be written in Scala, Java, or Python
- All data manipulation, aggregation, or ML applications must be implemented on top of Apache Spark
- Upload your **final code and resulting output files** to GitHub, or Bitbucket. If you don't have an account there, go ahead and create one for free
- Do not, however, post this document publicly, or make it available in your Git repo
- Your code should be clean, well-documented, and follow coding best-practices in line with the language of your choosing.

Git repo contents

We recommend that you organize your code and output files in the following manner:

```
.
..
src/   # Put all your code files in this directory.
out/   # This is where your output files go.
```

In terms of a file naming convention, we would ask that you make it obvious which task and part the files relates too, e.g.

```
src/task2_1.py
```

would be a Python solution for Part 2, Task 1.

Finally, these instructions are meant to be self-explanatory. In case there are doubts about a requirement, or what is being asked, make an assumption and explain the rationale behind your assumption in your code comments.

Part 1: Spark RDD API

In this part, we'll be leveraging Spark's RDD API to generate data summaries.

First, let's create a few summaries and counts on grocery shopping items. The dataset we're dealing with records shopping transactions of indivduals and can be accessed via the following URL:

```
https://raw.githubusercontent.com/stedy/Machine-Learning-with-R-datasets/master/groceries.csv
```

Task 1

Download the data file from the above location and make it accessible to Spark.

The first 5 rows will look like this:

```
[['citrus fruit', 'semi-finished bread', 'margarine', 'ready soups'],
 ['tropical fruit', 'yogurt', 'coffee'],
 ['whole milk'],
 ['pip fruit', 'yogurt', 'cream cheese ', 'meat spreads'],
 ['other vegetables',
  'whole milk',
  'condensed milk',
  'long life bakery product']]
```

Task 2

a.) Using Spark's **RDD API**, create a list of all (unique) products present in the transactions. Write out this list to a text file: `out/out_1_2a.txt`.

```
product a
product b
product c
```

b.) Again, using Spark only, write out the total count of products to a text file: `out/out_1_2b.txt` The contents of this file should look like this:

```
Count:
<XXX>
```

where `<XXX>` is the number of products.

Task 3

Create an RDD and using Spark APIs, determine the top 5 purchased products along with how often they were purchased (frequency count). Write the results out in descending order of frequency into a file `out/out_1_3.txt`.

As an example, this file could look like this:

```
('milk', 1100)
('butter', 1050)
('veg', 980)
...
```

Part 2: Spark Dataframe API

Now, let's turn our attention to **Spark's Dataframe API**. For this purpose, we will be leveraging a cleaned up, smaller version of the AirBnB data set.

Task 1

Download the parquet data file from this URL:

https://github.com/databricks/LearningSparkV2/blob/master/mlflow-project-example/data/sf-airbnb-clean.parquet (https://github.com/databricks/LearningSparkV2/blob/master/mlflow-project-example/data/sf-airbnb-clean.parquet)

and load it into a Spark data frame.

Task 2

Create CSV output file under `out/out_2_2.txt` that lists the minimum price, maximum price, and total row count from this data set. Use the following output column names in your resultant file:

```
min_price, max_price, row_count
```

Task 3

Calculate the average number of bathrooms and bedrooms across all the properties listed in this data set with a price of > 5000 and a review score being exactly equalt to 10.

Write the results into a CSV file `out/out_2_3.txt` with the following column headers:

```
avg_bathrooms, avg_bedrooms
```

Task 4

How many people can be accomodated by the property with the lowest price and highest rating?

Write the resulting number to a text file under `out/out_2_4.txt`.

Task 5

Using Apache Airflow's Dummy Operator, create an Airflow Dag that runs task 1, followed by tasks 2, and 3 in parallel, followed by tasks 4, 5, 6 all in parallel, in other words:

```
           Task 2     Task 4
Task 1 -->        --> Task 5
           Task 3     Task 6
```

It is not required to actually launch the applications; we just want to see what the DAG would look like if you were to submit those applications in the sequence prescribed above.

Save the DAG code into a file `src/task_2_5.py`

Part 3: Applied Machine Learning

For this section, your task will be to convert a small snippet of ML Python code implemented around `pandas` and `sklearn` into a functionally equivalent Spark version which may leverage Spark's ML or MLlib packages.

To simplify things, we have stripped down the ML code to a bare minimum.

Task 1

We'll be working with the famous Iris data set which can be fetched from UCI's machine learning repository in CSV format:

```
curl -L "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data" -o /tmp/iris.csv
```

Download this data set.

Problem description

The Data Science team has provided prototype code in Python which builds a basic, Logistic Regression model on the Iris data set.

We want to predict the type of flower - denoted as `class` in the CSV - based upon the 4 features which relate to certain flower measurements.

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression

df = pd.read_csv("/tmp/iris.csv",
                 names = ["sepal_length", "sepal_width", "petal_length", "petal_width", "class"])

df.head()

Out[58]:
```

	sepal_length	sepal_width	petal_length	petal_width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
# Separate features from class.
array = df.values
X = array[:,0:4]
y = array[:,4]
```

```
# Fit Logistic Regression classifier.
logreg = LogisticRegression(C=1e5)
logreg.fit(X, y)
```

```
Out[52]: LogisticRegression(C=100000.0, class_weight=None, dual=False,
                    fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                    max_iter=100, multi_class='auto', n_jobs=None, penalty='l2',
                    random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                    warm_start=False)
```

With this fitted model we can predict on the training data and convince ourselves that the model works.

```
# Predict on training data. Seems to work.
# 5.1     3.5     1.4     0.2     Iris-setosa
# 6.2     3.4     5.4     2.3     Iris-virginica

print(logreg.predict([[5.1, 3.5, 1.4, 0.2]]))
print(logreg.predict([[6.2, 3.4, 5.4, 2.3]]))

['Iris-setosa']
['Iris-virginica']
```

Task 2

Implement a Spark version of the above Python code and demonstrate that you can correctly predict on the training data, similar to what was done in the preceding section.

You should be able to run your model against the `pred_data` data frame in the code snippet:

```
pred_data = spark.createDataFrame(
    [(5.1, 3.5, 1.4, 0.2),
     (6.2, 3.4, 5.4, 2.3)],
    ["sepal_length", "sepal_width", "petal_length", "petal_width"])


predictions = <YOUR CODE>
```

Write out the prediction results to a CSV file `out/out_3_2.txt`:

```
class
<class 1>
<class 2
```

where `<class 1>` and `<class 2>` are your predicted classes for the two flowers. You can chose to write out numerical predictions or the actual names.