Big Data Fundamentals with PySpark (PySpark SQL & DataFrames)

RDD to DataFrame

Similar to RDDs, DataFrames are immutable and distributed data structures in Spark. Even though RDDs are a fundamental data structure in Spark, working with data in DataFrame is easier than RDD most of the time and so understanding of how to convert RDD to DataFrame is necessary.

In this exercise, you'll first make an RDD using the `sample_list` which contains the list of tuples `('Mona',20), ('Jennifer',34),('John',20), ('Jim',26)` with each tuple contains the name of the person and their age. Next, you'll create a DataFrame using the RDD and the schema (which is the list of 'Name' and 'Age') and finally confirm the output as PySpark DataFrame.

Remember, you already have a SparkContext `sc` and SparkSession `spark` available in your workspace.

```python
# Create a list of tuples
sample_list = [('Mona',20),('Jennifer',34), ('John',20), ('Jim',26)]

# Create a RDD from the list
rdd = sc.parallelize(sample_list)

# Create a PySpark DataFrame
names_df = spark.createDataFrame(rdd, schema=['Name', 'Age'])

# Check the type of names_df
print("The type of names_df is", type(names_df))
```

Loading CSV into DataFrame

In the previous exercise, you have seen a method of creating DataFrame but generally, loading data from CSV file is the most common method of creating DataFrames. In this exercise, you'll create a PySpark DataFrame from a `people.csv` file that is already provided to you as a `file_path` and confirm the created object is a PySpark DataFrame.

Remember, you already have SparkSession `spark` and `file_path` variable (which is the path to the `people.csv` file) available in your workspace.

```python
# Create an DataFrame from file_path
people_df = spark.read.csv(file_path, header=True, inferSchema=True)

# Check the type of people_df
print("The type of people_df is", type(people_df))
```

Inspecting data in PySpark DataFrame

Inspecting data is very crucial before performing analysis such as plotting, modeling, training etc., In this simple exercise, you'll inspect the data in the `people_df` DataFrame that you have created in the previous exercise using basic DataFrame operators.

Remember, you already have SparkSession `spark` and `people_df` DataFrame available in your workspace.

```python
# Print the first 10 observations
people_df.show(10)

# Count the number of rows
print("There are {} rows in the people_df DataFrame.".format(people_df.count()))

# Count the number of columns and their names
print("There are {} columns in the people_df DataFrame and their names are
{}".format(len(people_df.columns), people_df.columns))
```

PySpark DataFrame subsetting and cleaning

After data inspection, it is often necessary to clean the data which mainly involves subsetting, renaming the columns, removing duplicated rows etc., PySpark DataFrame API provides several operators to do this. In this exercise, your job is to subset 'name', 'sex' and 'date of birth' columns from `people_df` DataFrame, remove any duplicate rows from that dataset and count the number of rows before and after duplicates removal step.

Remember, you already have SparkSession `spark` and `people_df` DataFrames available in your workspace.

```python
# Select name, sex and date of birth columns
people_df_sub = people_df.select('name', 'sex', 'date of birth')

# Print the first 10 observations from people_df_sub
```

```python
people_df_sub.show(10)

# Remove duplicate entries from people_df_sub
people_df_sub_nodup = people_df_sub.dropDuplicates()

# Count the number of rows
print("There were {} rows before removing duplicates, and {} rows after removing duplicates".format(people_df_sub.count(), people_df_sub_nodup.count()))
```

Filtering your DataFrame

In the previous exercise, you have subset the data using `select()` operator which is mainly used to subset the DataFrame column-wise. What if you want to subset the DataFrame based on a condition (for example, select all rows where the sex is Female). In this exercise, you will filter the rows in the `people_df` DataFrame in which 'sex' is female and male and create two different datasets. Finally, you'll count the number of rows in each of those datasets.

Remember, you already have SparkSession `spark` and `people_df` DataFrame available in your workspace.

```python
# Filter people_df to select females
people_df_female = people_df.filter(people_df.sex == "female")

# Filter people_df to select males
people_df_male = people_df.filter(people_df.sex == "male")

# Count the number of rows
print("There are {} rows in the people_df_female DataFrame and {} rows in the people_df_male DataFrame".format(people_df_female.count(), people_df_male.count()))
```

Running SQL Queries Programmatically

DataFrames can easily be manipulated using SQL queries in PySpark. The `sql()` function on a SparkSession enables applications to run SQL queries programmatically and returns the result as another DataFrame. In this exercise, you'll create a temporary table of the `people_df` DataFrame that you created previously, then construct a query to select the names of the people from the temporary table and assign the result to a new DataFrame.

Remember, you already have SparkSession `spark` and `people_df` DataFrame available in your workspace.

```python
# Create a temporary table "people"
people_df.createOrReplaceTempView("people")

```

```python
# Construct a query to select the names of the people from the temporary table "people"
query = '''SELECT name FROM people'''

# Assign the result of Spark's query to people_df_names
people_df_names = spark.sql(query)

# Print the top 10 names of the people
people_df_names.show(10)
```

SQL queries for filtering Table

In the previous exercise, you have run a simple SQL query on a DataFrame. There are more sophisticated queries you can construct to obtain the result that you want and use it for downstream analysis such as data visualization and Machine Learning. In this exercise, we will use the temporary table `people` that you created previously and filter out the rows where the "sex" is male and female and create two DataFrames.

Remember, you already have SparkSession `spark` and `people` temporary table available in your workspace.

```python
# Filter the people table to select female sex
people_female_df = spark.sql('SELECT * FROM people WHERE sex=="female"')

# Filter the people table DataFrame to select male sex
people_male_df = spark.sql('SELECT * FROM people WHERE sex=="male"')

# Count the number of rows in both DataFrames
print("There are {} rows in the people_female_df and {} rows in the people_male_df
DataFrames".format(people_female_df.count(), people_male_df.count()))
```

PySpark DataFrame Visualization

Graphical representations or visualization of data is imperative for understanding as well as interpreting the data. In this simple data visualization exercise, you'll first print the column names of `names_df` DataFrame that you created earlier, then convert the `names_df` to Pandas DataFrame and finally plot the contents as horizontal bar plot with names of the people on the x-axis and their age on the y-axis.

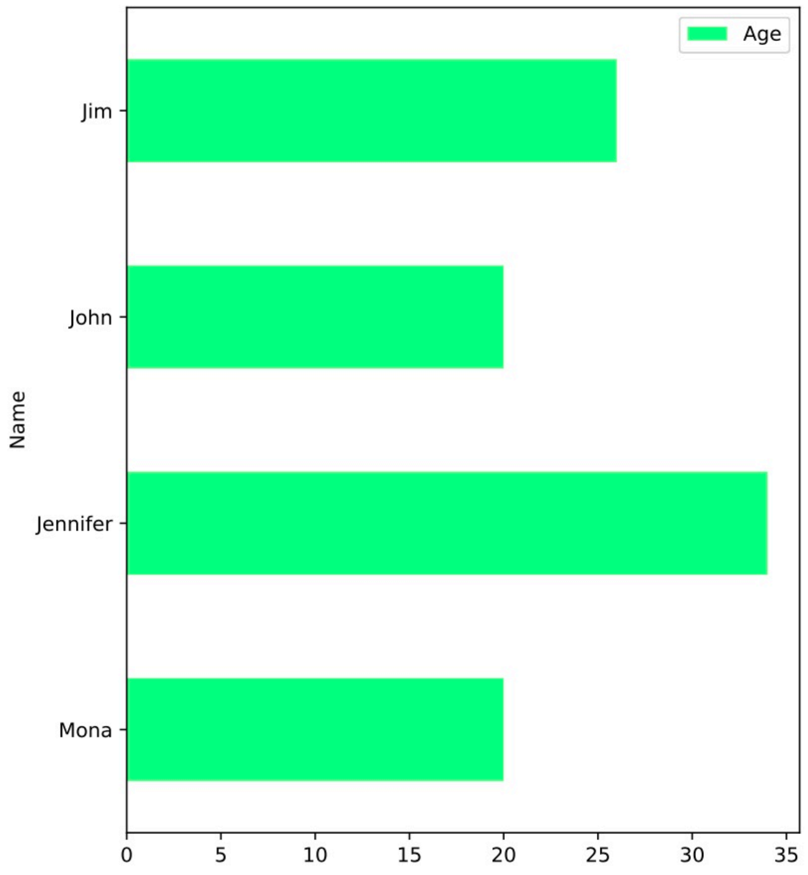
Remember, you already have SparkSession `spark` and `names_df` DataFrame available in your workspace.

```python
# Check the column names of names_df
print("The column names of names_df are", names_df.columns)

```

```
# Convert to Pandas DataFrame
df_pandas = names_df.toPandas()

# Create a horizontal bar plot
df_pandas.plot(kind='barh', x='Name', y='Age', colormap='winter_r')
plt.show()
```



Create a DataFrame from CSV file

Every 4 years, the soccer fans throughout the world celebrates a festival called “Fifa World Cup” and with that, everything seems to change in many countries. In this 3 part exercise, you'll be doing some exploratory data analysis (EDA) on the "FIFA 2018 World Cup Player" dataset using PySpark SQL which involve DataFrame operations, SQL queries and visualization.

In the first part, you'll load FIFA 2018 World Cup Players dataset (`Fifa2018_dataset.csv`) which is in CSV format into a PySpark's dataFrame and inspect the data using basic DataFrame operations.

Remember, you already have SparkSession `spark` and `file_path` variable (which is the path to the `Fifa2018_dataset.csv` file) available in your workspace.

```
# Load the Dataframe
```

```python
fifa_df = spark.read.csv(file_path, header=True, inferSchema=True)

# Check the schema of columns
fifa_df.printSchema()

# Show the first 10 observations
fifa_df.show(10)

# Print the total number of rows
print("There are {} rows in the fifa_df DataFrame".format(fifa_df.count()))
```

SQL Queries on DataFrame

The `fifa_df` DataFrame that we created has additional information about datatypes and names of columns associated with it. This additional information allows PySpark SQL to run SQL queries on DataFrame. SQL queries are concise and easy to run compared to DataFrame operations. But in order to apply SQL queries on DataFrame first, you need to create a temporary view of DataFrame as a table and then apply SQL queries on the created table (Running SQL Queries Programmatically).

In the second part, you'll create a temporary table of `fifa_df` DataFrame and run SQL queries to extract the 'Age' column of players from Germany.

You already have a SparkContext `spark` and `fifa_df` available in your workspace.

```python
# Create a temporary view of fifa_df
fifa_df.createOrReplaceTempView('fifa_df_table')

# Construct the "query"
query = '''SELECT Age FROM fifa_df_table WHERE Nationality == "Germany"'''

# Apply the SQL "query"
fifa_df_germany_age = spark.sql(query)

# Generate basic statistics
fifa_df_germany_age.describe().show()
```

Data Visualization

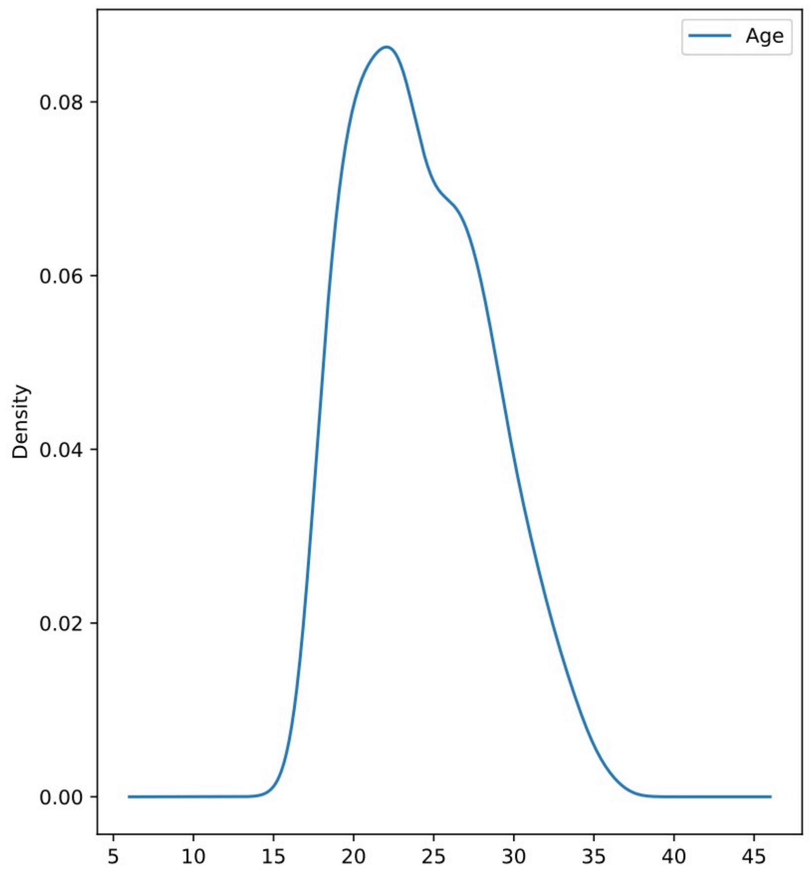
Data visualization is important for exploratory data analysis (EDA). PySpark DataFrame is a perfect for data visualization compared to RDDs because of its inherent structure and schema.

In this third part, you'll create a histogram of the ages of all the players from Germany from the DataFrame that you created in the previous exercise. For this, you'll first convert the PySpark DataFrame into Pandas DataFrame and use matplotlib's `plot()` function to create a density plot of ages of all players from Germany.

Remember, you already have SparkSession `spark` , `fifa_df_table` temporary table and `fifa_df_germany_age` DataFrame available in your workspace.

```python
# Convert fifa_df to fifa_df_germany_age_pandas DataFrame
fifa_df_germany_age_pandas = fifa_df_germany_age.toPandas()

# Plot the 'Age' density of Germany Players
fifa_df_germany_age_pandas.plot(kind='density')
plt.show()
```


Age
Density