Analysis of Electricity Demand, Consumption, and Electric Vehicle Trends Across Regions from 2016 to 2021

Processing Big Data for Analytics Applications
Fall 2023

Cleaning

SSH-in-browser UPLOAD FILE DOWNLOAD FILE

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.1.2
      /_/

Using Scala version 2.12.14 (OpenJDK 64-Bit Server VM, Java 1.8.0_322)
Type in expressions to have them evaluated.
Type :help for more information.

scala> import org.apache.spark.sql.{DataFrame, SparkSession}
import org.apache.spark.sql.{DataFrame, SparkSession}

scala> import org.apache.spark.sql.functions._
import org.apache.spark.sql.functions._

scala> import org.apache.spark.sql.types.DoubleType
import org.apache.spark.sql.types.DoubleType

scala> import org.apache.spark.rdd.RDD
import org.apache.spark.rdd.RDD

scala>

scala> // Initialize Spark Session

scala> val spark = SparkSession.builder.appName("Data Cleaning for Electricity Rates").getOrCreate()
23/12/03 17:52:48 WARN org.apache.spark.sql.SparkSession$Builder: Using an existing SparkSession; some spark core configurations may not take effect.
spark: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@4218b6d3

scala> import spark.implicits._
import spark.implicits._

scala>

scala> // Define the state-to-region mapping

scala> val stateToRegion = Map(
     |   "WA" -> "NW", "CA" -> "CAL", "UT" -> "SW", "VA" -> "SE", "OR" -> "NW", "TX" -> "TEX", "MI" -> "MIDW",
     |   "TN" -> "TEN", "DE" -> "MIDA", "MN" -> "MIDW", "HI" -> "SW", "ID" -> "NW", "AL" -> "SE", "IN" -> "MIDW",
     |   "WY" -> "NW", "GA" -> "SE", "FL" -> "FLA", "KY" -> "SE", "DC" -> "MIDA", "MD" -> "MIDA", "AZ" -> "SW",
     |   "NE" -> "MIDW", "LA" -> "SE", "PA" -> "NE", "NC" -> "CAR", "NY" -> "NY", "SD" -> "MIDW", "AK" -> "NW",
     |   "NM" -> "SW", "MO" -> "MIDW", "CO" -> "SW", "IL" -> "MIDW", "NV" -> "SW", "SC" -> "CAR", "MS" -> "SE",
```

SSH-in-browser UPLOAD FILE DOWNLOAD FILE

Download file

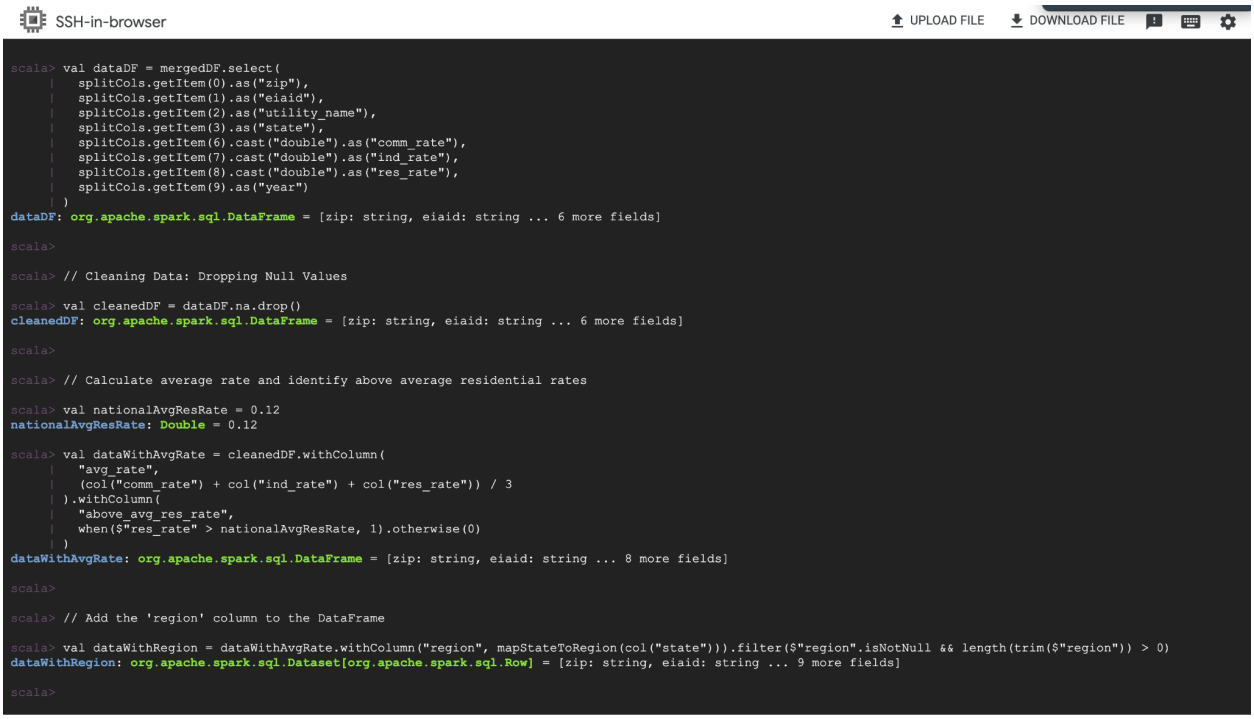
```
scala> val stateToRegion = Map(
     |   "WA" -> "NW", "CA" -> "CAL", "UT" -> "SW", "VA" -> "SE", "OR" -> "NW", "TX" -> "TEX", "MI" -> "MIDW",
     |   "TN" -> "TEN", "DE" -> "MIDA", "MN" -> "MIDW", "HI" -> "SW", "ID" -> "NW", "AL" -> "SE", "IN" -> "MIDW",
     |   "WY" -> "NW", "GA" -> "SE", "FL" -> "FLA", "KY" -> "SE", "DC" -> "MIDA", "MD" -> "MIDA", "AZ" -> "SW",
     |   "NE" -> "MIDW", "LA" -> "SE", "PA" -> "NE", "NC" -> "CAR", "NY" -> "NY", "SD" -> "MIDW", "AK" -> "NW",
     |   "NM" -> "SW", "MO" -> "MIDW", "CO" -> "SW", "IL" -> "MIDW", "NV" -> "SW", "SC" -> "CAR", "MS" -> "SE",
     |   "MT" -> "NW", "ND" -> "MIDW", "OH" -> "MIDW", "MA" -> "NE", "CT" -> "NE", "NJ" -> "NE", "WI" -> "MIDW",
     |   "IA" -> "MIDW", "KS" -> "CENT", "OK" -> "SW", "AR" -> "SE", "NH" -> "NE", "ME" -> "NE"
     | )
stateToRegion: scala.collection.immutable.Map[String,String] = Map(MA -> NE, IN -> MIDW, ID -> NW, NM -> SW, OR -> NW, IA -> MIDW, IL -> MIDW, TN -> TEN, MO -> MID
W, ME -> NE, AZ -> SW, AK -> NW, WA -> NW, SD -> MIDW, KY -> SE, NJ -> NE, TX -> TEX, MI -> MIDW, MD -> MIDA, NV -> SW, NE -> MIDW, MN -> MIDW, KS -> CENT, OK -> S
W, CT -> NE, OH -> MIDW, AR -> SE, FL -> FLA, WI -> MIDW, CO -> SW, MT -> NW, DC -> MIDA, ND -> MIDW, PA -> NE, GA -> SE, NH -> NE, HI -> SW, WY -> NW, LA -> SE, C
A -> CAL, UT -> SW, AL -> SE, VA -> SE, NC -> CAR, NY -> NY, SC -> CAR, MS -> SE, DE -> MIDA)

scala>

scala> // Function to map state to region

scala> val mapStateToRegion = udf((state: String) => stateToRegion.getOrElse(state, null))
mapStateToRegion: org.apache.spark.sql.expressions.UserDefinedFunction = SparkUserDefinedFunction($Lambda$2713/401992365@3966e420,StringType,List(Some(class[value[
0]: string])),Some(class[value[0]: string]),None,true,true)

scala>

scala> // Function to read CSV, add year, and optionally skip the first row

scala> def readCsvWithYearAndSkipFirstRow(path: String, year: Int, skipFirstRow: Boolean): RDD[String] = {
     |   val rdd = spark.sparkContext.textFile(path).mapPartitionsWithIndex { (idx, iter) =>
     |     if (idx == 0 && skipFirstRow) iter.drop(1) else iter
     |   }
     |   rdd.map(row => s"$row,$year")
     | }
readCsvWithYearAndSkipFirstRow: (path: String, year: Int, skipFirstRow: Boolean)org.apache.spark.rdd.RDD[String]

scala>

scala> // Define paths to your CSV files

scala> val basePath = "/user/ar7165_nyu_edu/HW8"
basePath: String = /user/ar7165_nyu_edu/HW8

scala> val path2016 = s"$basePath/electricity_rates_2016.csv"
path2016: String = /user/ar7165_nyu_edu/HW8/electricity_rates_2016.csv
```

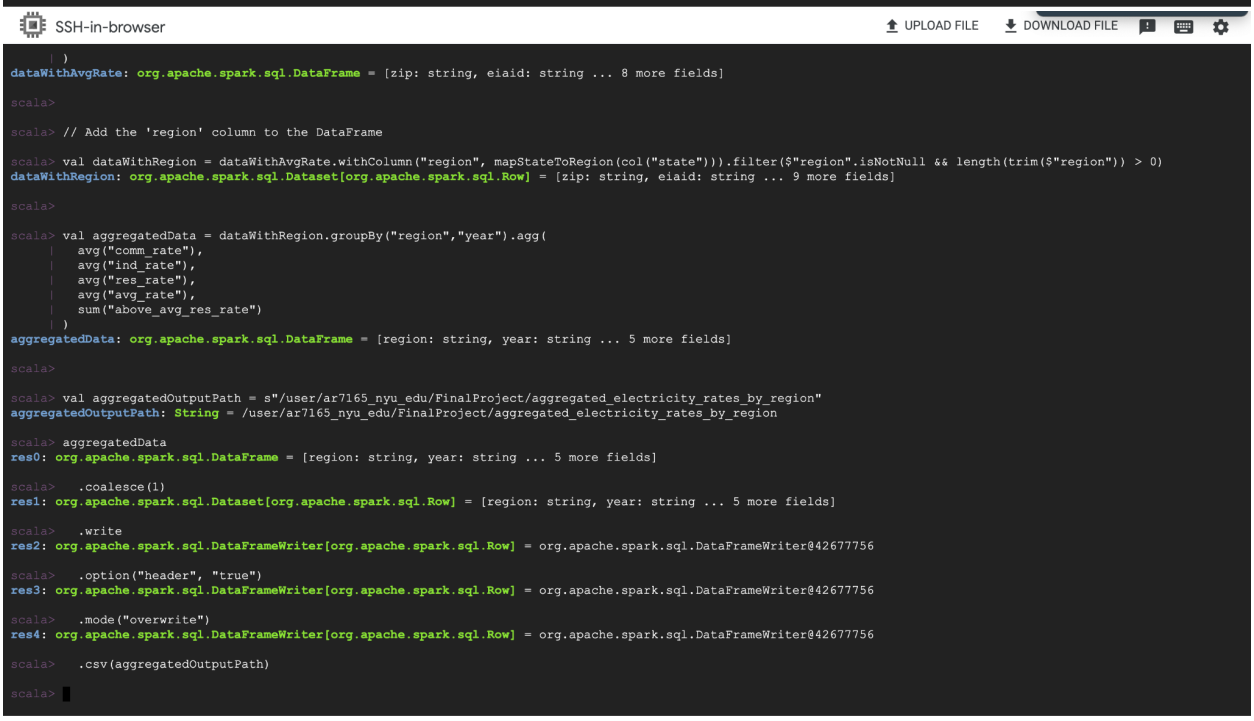
SSH-in-browser UPLOAD FILE DOWNLOAD FILE

```
scala> val path2016 = s"$basePath/electricity_rates_2016.csv"
path2016: String = /user/ar7165_nyu_edu/HW8/electricity_rates_2016.csv

scala> val path2017 = s"$basePath/electricity_rates_2017.csv"
path2017: String = /user/ar7165_nyu_edu/HW8/electricity_rates_2017.csv

scala> val path2018 = s"$basePath/electricity_rates_2018.csv"
path2018: String = /user/ar7165_nyu_edu/HW8/electricity_rates_2018.csv

scala> val path2019 = s"$basePath/electricity_rates_2019.csv"
path2019: String = /user/ar7165_nyu_edu/HW8/electricity_rates_2019.csv

scala> val path2020 = s"$basePath/electricity_rates_2020.csv"
path2020: String = /user/ar7165_nyu_edu/HW8/electricity_rates_2020.csv

scala> val path2021 = s"$basePath/electricity_rates_2021.csv"
path2021: String = /user/ar7165_nyu_edu/HW8/electricity_rates_2021.csv

scala>

scala> // Read CSV files and add the year column

scala> val rdd2016 = readCsvWithYearAndSkipFirstRow(path2016, 2016, skipFirstRow = false)
rdd2016: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[3] at map at <console>:36

scala> val rdd2017 = readCsvWithYearAndSkipFirstRow(path2017, 2017, skipFirstRow = true)
rdd2017: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[7] at map at <console>:36

scala> val rdd2018 = readCsvWithYearAndSkipFirstRow(path2018, 2018, skipFirstRow = true)
rdd2018: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[11] at map at <console>:36

scala> val rdd2019 = readCsvWithYearAndSkipFirstRow(path2019, 2019, skipFirstRow = true)
rdd2019: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[15] at map at <console>:36

scala> val rdd2020 = readCsvWithYearAndSkipFirstRow(path2020, 2020, skipFirstRow = true)
rdd2020: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[19] at map at <console>:36

scala> val rdd2021 = readCsvWithYearAndSkipFirstRow(path2021, 2021, skipFirstRow = true)
rdd2021: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[23] at map at <console>:36

scala>

scala> // Merge RDDs
```

SSH-in-browser UPLOAD FILE DOWNLOAD FILE

```
scala> // Merge RDDs

scala> val mergedRDD = rdd2016.union(rdd2017).union(rdd2018).union(rdd2019).union(rdd2020).union(rdd2021)
23/12/03 17:52:54 WARN org.apache.hadoop.util.concurrent.ExecutorHelper: Thread (Thread[GetFileInfo #0,5,main]) interrupted:
java.lang.InterruptedException
        at com.google.common.util.concurrent.AbstractFuture.get(AbstractFuture.java:510)
        at com.google.common.util.concurrent.FluentFuture$TrustedFuture.get(FluentFuture.java:88)
        at org.apache.hadoop.util.concurrent.ExecutorHelper.logThrowableFromAfterExecute(ExecutorHelper.java:48)
        at org.apache.hadoop.util.concurrent.HadoopThreadPoolExecutor.afterExecute(HadoopThreadPoolExecutor.java:90)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1157)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
        at java.lang.Thread.run(Thread.java:750)
23/12/03 17:52:54 WARN org.apache.hadoop.util.concurrent.ExecutorHelper: Thread (Thread[GetFileInfo #1,5,main]) interrupted:
java.lang.InterruptedException
        at com.google.common.util.concurrent.AbstractFuture.get(AbstractFuture.java:510)
        at com.google.common.util.concurrent.FluentFuture$TrustedFuture.get(FluentFuture.java:88)
        at org.apache.hadoop.util.concurrent.ExecutorHelper.logThrowableFromAfterExecute(ExecutorHelper.java:48)
        at org.apache.hadoop.util.concurrent.HadoopThreadPoolExecutor.afterExecute(HadoopThreadPoolExecutor.java:90)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1157)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
        at java.lang.Thread.run(Thread.java:750)
mergedRDD: org.apache.spark.rdd.RDD[String] = UnionRDD[28] at union at <console>:38

scala>

scala> // Convert RDD to DataFrame and split columns

scala> val mergedDF = mergedRDD.toDF("value")
mergedDF: org.apache.spark.sql.DataFrame = [value: string]

scala> val splitCols = split($"value", ",")
splitCols: org.apache.spark.sql.Column = split(value, ,, -1)

scala>

scala> val dataDF = mergedDF.select(
     |   splitCols.getItem(0).as("zip"),
     |   splitCols.getItem(1).as("eiaid"),
     |   splitCols.getItem(2).as("utility_name"),
     |   splitCols.getItem(3).as("state"),
     |   splitCols.getItem(6).cast("double").as("comm_rate"),
     |   splitCols.getItem(7).cast("double").as("ind_rate"),
     |   splitCols.getItem(8).cast("double").as("res_rate"),
     |   splitCols.getItem(9).as("year")
```





This Cleaning script processes and analyzes electricity rate data from 2016 to 2021. It initializes a Spark session and defines a function to read CSV files, adding a year column and optionally skipping the first row. The script also includes a mapping of U.S. states to their corresponding regions. Electricity rate data for each year from 2016 to 2021 is read from separate CSV files, merged into a single RDD, and then converted into a DataFrame. The DataFrame is split into columns and appropriately cast to relevant data types. The script performs data cleaning by dropping rows with null values and calculates an average rate for each record. It then identifies

records with above-average residential rates based on a predefined national average. A 'region' column is added to the Data Frame using the state-to-region mapping. The data is aggregated by region and year to compute average rates and the sum of instances with above-average residential rates. Finally, the aggregated data is coalesced into a single file and written to a CSV file, providing a structured and clean dataset for further analysis or reporting.

Profiling

SSH-in-browser UPLOAD FILE DOWNLOAD FILE

```
scala> import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.SparkSession

scala> import org.apache.spark.sql.functions._
import org.apache.spark.sql.functions._

scala> import org.apache.spark.sql.types._
import org.apache.spark.sql.types._

scala> import org.apache.spark.rdd.RDD
import org.apache.spark.rdd.RDD

scala> import org.apache.spark.ml.feature.Imputer
import org.apache.spark.ml.feature.Imputer

scala>

scala>

scala> val spark = SparkSession.builder.appName("Electricity Rates Profiling").getOrCreate()
23/12/03 18:45:52 WARN org.apache.spark.sql.SparkSession$Builder: Using an existing SparkSession; some spark core configurations may not take effect.
spark: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@4b6e9c75

scala>

scala> // Function to read CSV and add year column

scala> def readCsvWithYearAndSkipFirstRow(path: String, year: Int, skipFirstRow: Boolean): RDD[String] = {
     |   val rdd = spark.sparkContext.textFile(path).mapPartitionsWithIndex { (idx, iter) =>
     |     if (idx == 0 && skipFirstRow) iter.drop(1) else iter
     |   }
     |   rdd.map(row => s"$row,$year")
     | }
readCsvWithYearAndSkipFirstRow: (path: String, year: Int, skipFirstRow: Boolean)org.apache.spark.rdd.RDD[String]

scala>

scala> // Define paths to your CSV files

scala> val basePath = "/user/ar7165_nyu_edu/HW8"
basePath: String = /user/ar7165_nyu_edu/HW8

scala> val path2016 = s"$basePath/electricity_rates_2016.csv"
```

SSH-in-browser UPLOAD FILE DOWNLOAD FILE

```
scala> val path2016 = s"$basePath/electricity_rates_2016.csv"
path2016: String = /user/ar7165_nyu_edu/HW8/electricity_rates_2016.csv

scala> val path2017 = s"$basePath/electricity_rates_2017.csv"
path2017: String = /user/ar7165_nyu_edu/HW8/electricity_rates_2017.csv

scala> val path2018 = s"$basePath/electricity_rates_2018.csv"
path2018: String = /user/ar7165_nyu_edu/HW8/electricity_rates_2018.csv

scala> val path2019 = s"$basePath/electricity_rates_2019.csv"
path2019: String = /user/ar7165_nyu_edu/HW8/electricity_rates_2019.csv

scala> val path2020 = s"$basePath/electricity_rates_2020.csv"
path2020: String = /user/ar7165_nyu_edu/HW8/electricity_rates_2020.csv

scala> val path2021 = s"$basePath/electricity_rates_2021.csv"
path2021: String = /user/ar7165_nyu_edu/HW8/electricity_rates_2021.csv

scala>

scala> // Read CSV files and add the year column

scala> val rdd2016 = readCsvWithYearAndSkipFirstRow(path2016, 2016, skipFirstRow = false)
rdd2016: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[86] at map at <console>:44

scala> val rdd2017 = readCsvWithYearAndSkipFirstRow(path2017, 2017, skipFirstRow = true)
rdd2017: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[90] at map at <console>:44

scala> val rdd2018 = readCsvWithYearAndSkipFirstRow(path2018, 2018, skipFirstRow = true)
rdd2018: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[94] at map at <console>:44

scala> val rdd2019 = readCsvWithYearAndSkipFirstRow(path2019, 2019, skipFirstRow = true)
rdd2019: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[98] at map at <console>:44

scala> val rdd2020 = readCsvWithYearAndSkipFirstRow(path2020, 2020, skipFirstRow = true)
rdd2020: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[102] at map at <console>:44

scala> val rdd2021 = readCsvWithYearAndSkipFirstRow(path2021, 2021, skipFirstRow = true)
rdd2021: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[106] at map at <console>:44

scala>

scala> // Merge RDDs
```

SSH-in-browser UPLOAD FILE DOWNLOAD FILE

```
scala> // Merge RDDs

scala> val mergedRDD = rdd2016.union(rdd2017).union(rdd2018).union(rdd2019).union(rdd2020).union(rdd2021)
mergedRDD: org.apache.spark.rdd.RDD[String] = UnionRDD[111] at union at <console>:45

scala>

scala> // Convert RDD to DataFrame and split columns

scala> val mergedDF = mergedRDD.toDF("value")
mergedDF: org.apache.spark.sql.DataFrame = [value: string]

scala> val splitCols = split($"value", ",")
splitCols: org.apache.spark.sql.Column = split(value, ,, -1)

scala>

scala> val dataDF = mergedDF.select(
     |   splitCols.getItem(0).as("zip"),
     |   splitCols.getItem(1).as("eiaid"),
     |   splitCols.getItem(2).as("utility_name"),
     |   splitCols.getItem(3).as("state"),
     |   splitCols.getItem(6).cast("double").as("comm_rate"),
     |   splitCols.getItem(7).cast("double").as("ind_rate"),
     |   splitCols.getItem(8).cast("double").as("res_rate"),
     |   splitCols.getItem(9).as("year")
     | )
dataDF: org.apache.spark.sql.DataFrame = [zip: string, eiaid: string ... 6 more fields]

scala>

scala> // Data Profiling

scala> // Show the schema of the DataFrame

scala> dataDF.printSchema()
root
 |-- zip: string (nullable = true)
 |-- eiaid: string (nullable = true)
 |-- utility_name: string (nullable = true)
 |-- state: string (nullable = true)
 |-- comm_rate: double (nullable = true)
 |-- ind_rate: double (nullable = true)
 |-- res_rate: double (nullable = true)
```

SSH-in-browser UPLOAD FILE DOWNLOAD FILE

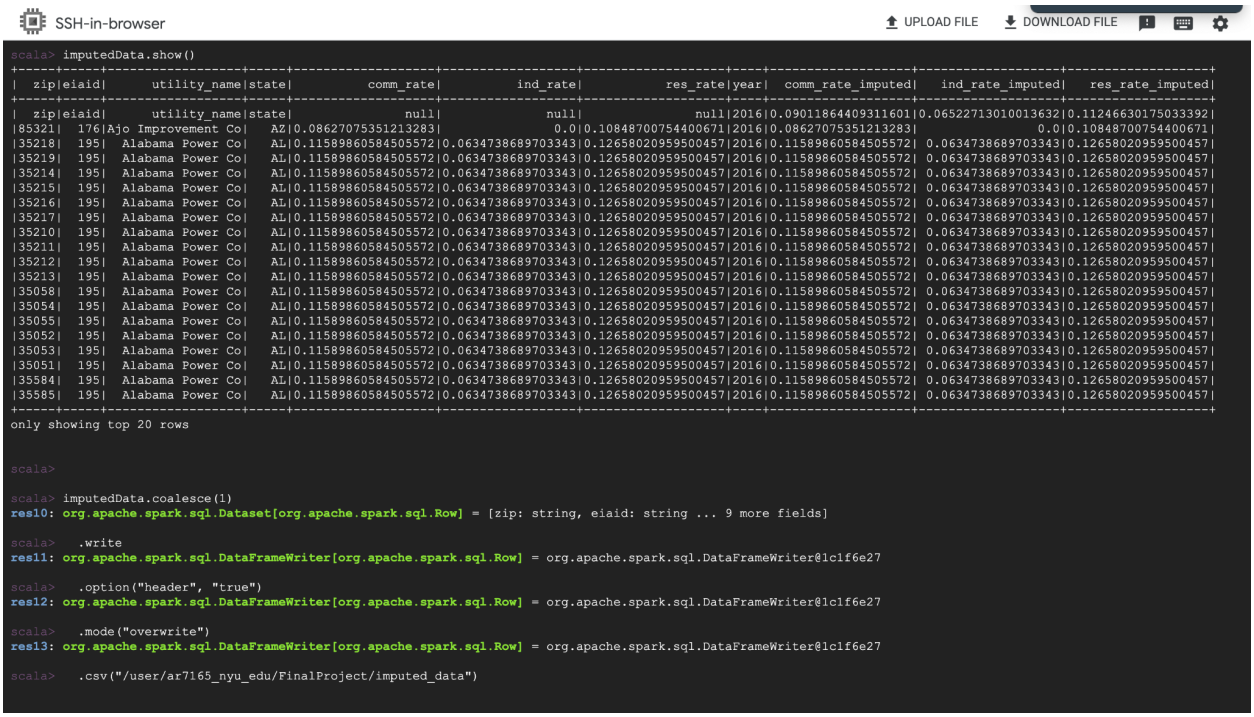
```
scala> dataDF.printSchema()
root
 |-- zip: string (nullable = true)
 |-- eiaid: string (nullable = true)
 |-- utility_name: string (nullable = true)
 |-- state: string (nullable = true)
 |-- comm_rate: double (nullable = true)
 |-- ind_rate: double (nullable = true)
 |-- res_rate: double (nullable = true)
 |-- year: string (nullable = true)


scala>

scala> // Show the number of rows in the DataFrame

scala> val numRows = dataDF.count()
numRows: Long = 313870

scala> println(s"Number of rows: $numRows")
Number of rows: 313870

scala>

scala> // Check for null values in each column

scala> // Check for null values in each column

scala> dataDF.columns.foreach { colName =>
     |   val nullCount = dataDF.select(count(when(col(colName).isNull || isnan(col(colName)), true)).alias(s"Nulls_in_$colName"))
     |   nullCount.show()
     | }
+------------+
|Nulls_in_zip|
+------------+
|           0|
+------------+

+--------------+
|Nulls_in_eiaid|
+--------------+
|             0|
+--------------+
```

SSH-in-browser UPLOAD FILE DOWNLOAD FILE

```
+--------------+
|Nulls_in_eiaid|
+--------------+
|             0|
+--------------+

+---------------------+
|Nulls_in_utility_name|
+---------------------+
|                    0|
+---------------------+

+--------------+
|Nulls_in_state|
+--------------+
|             0|
+--------------+

+------------------+
|Nulls_in_comm_rate|
+------------------+
|             14058|
+------------------+

+-----------------+
|Nulls_in_ind_rate|
+-----------------+
|                1|
+-----------------+

+-----------------+
|Nulls_in_res_rate|
+-----------------+
|                1|
+-----------------+

+-------------+
|Nulls_in_year|
+-------------+
|            0|
+-------------+


scala>
```

SSH-in-browser UPLOAD FILE DOWNLOAD FILE

```
scala>

scala> // Instantiate the Imputer transformer to fill in missing values with the mean

scala> val columnsToImpute = Array("comm_rate", "ind_rate", "res_rate")
columnsToImpute: Array[String] = Array(comm_rate, ind_rate, res_rate)

scala> val imputer = new Imputer()
imputer: org.apache.spark.ml.feature.Imputer = imputer_f5f99fb6e3ba

scala>   .setInputCols(columnsToImpute)
res6: imputer.type = imputer_f5f99fb6e3ba

scala>   .setOutputCols(columnsToImpute.map(c => s"${c}_imputed"))
res7: res6.type = imputer_f5f99fb6e3ba

scala>   .setStrategy("mean") // Use the mean value of the column for imputation
res8: res7.type = imputer_f5f99fb6e3ba

scala>

scala> // Apply the Imputer transformer to the DataFrame with null values

scala> val imputedData = imputer.fit(dataDF).transform(dataDF)
imputedData: org.apache.spark.sql.DataFrame = [zip: string, eiaid: string ... 9 more fields]

scala>

scala> // Show the result with imputed values

scala> imputedData.show()
+-----+-----+--------------------+-----+--------------------+--------------------+--------------------+----+--------------------+--------------------+--------------------+
|  zip|eiaid|      utility_name|state|          comm_rate|          ind_rate|           res_rate|year|  comm_rate_imputed|   ind_rate_imputed|   res_rate_imputed|
+-----+-----+--------------------+-----+--------------------+--------------------+--------------------+----+--------------------+--------------------+--------------------+
|  zip|eiaid|      utility_name|state|                null|                null|                null|2016|0.09011864409311601|0.06522713010013632|0.11246630175033392|
|85321|  176|Ajo Improvement Co|   AZ|0.08627075351213283|                 0.0|0.10848700754400671|2016|0.08627075351213283|                 0.0|0.10848700754400671|
|35218|  195|  Alabama Power Co|   AL|0.11589860584505572|0.06347386897033433|0.12658020959500457|2016|0.11589860584505572| 0.06347386897033433|0.12658020959500457|
|35219|  195|  Alabama Power Co|   AL|0.11589860584505572|0.06347386897033433|0.12658020959500457|2016|0.11589860584505572| 0.06347386897033433|0.12658020959500457|
|35214|  195|  Alabama Power Co|   AL|0.11589860584505572|0.06347386897033433|0.12658020959500457|2016|0.11589860584505572| 0.06347386897033433|0.12658020959500457|
|35215|  195|  Alabama Power Co|   AL|0.11589860584505572|0.06347386897033433|0.12658020959500457|2016|0.11589860584505572| 0.06347386897033433|0.12658020959500457|
|35216|  195|  Alabama Power Co|   AL|0.11589860584505572|0.06347386897033433|0.12658020959500457|2016|0.11589860584505572| 0.06347386897033433|0.12658020959500457|
|35217|  195|  Alabama Power Co|   AL|0.11589860584505572|0.06347386897033433|0.12658020959500457|2016|0.11589860584505572| 0.06347386897033433|0.12658020959500457|
|35210|  195|  Alabama Power Co|   AL|0.11589860584505572|0.06347386897033433|0.12658020959500457|2016|0.11589860584505572| 0.06347386897033433|0.12658020959500457|
|35211|  195|  Alabama Power Co|   AL|0.11589860584505572|0.06347386897033433|0.12658020959500457|2016|0.11589860584505572| 0.06347386897033433|0.12658020959500457|
```




```
scala> imputedData.show()
+-----+-----+-----------------+-----+-------------------+-------------------+-------------------+----+-------------------+-------------------+-------------------+
|  zip|eiaid|      utility_name|state|          comm_rate|           ind_rate|           res_rate|year|  comm_rate_imputed|   ind_rate_imputed|   res_rate_imputed|
+-----+-----+-----------------+-----+-------------------+-------------------+-------------------+----+-------------------+-------------------+-------------------+
|  zip|eiaid|      utility_name|state|               null|               null|               null|2016|0.09011864409311601|0.06522713010013632|0.11246630175033392|
|85321|  176|Ajo Improvement Co|   AZ|0.08627075351213283|                0.0|0.10848700754400671|2016|0.08627075351213283|                0.0|0.10848700754400671|
|35218|  195|  Alabama Power Co|   AL|0.11589860584505572|0.06347386897033343|0.12658020959500457|2016|0.11589860584505572| 0.06347386897033343|0.12658020959500457|
|35219|  195|  Alabama Power Co|   AL|0.11589860584505572|0.06347386897033343|0.12658020959500457|2016|0.11589860584505572| 0.06347386897033343|0.12658020959500457|
|35214|  195|  Alabama Power Co|   AL|0.11589860584505572|0.06347386897033343|0.12658020959500457|2016|0.11589860584505572| 0.06347386897033343|0.12658020959500457|
|35215|  195|  Alabama Power Co|   AL|0.11589860584505572|0.06347386897033343|0.12658020959500457|2016|0.11589860584505572| 0.06347386897033343|0.12658020959500457|
|35216|  195|  Alabama Power Co|   AL|0.11589860584505572|0.06347386897033343|0.12658020959500457|2016|0.11589860584505572| 0.06347386897033343|0.12658020959500457|
|35217|  195|  Alabama Power Co|   AL|0.11589860584505572|0.06347386897033343|0.12658020959500457|2016|0.11589860584505572| 0.06347386897033343|0.12658020959500457|
|35210|  195|  Alabama Power Co|   AL|0.11589860584505572|0.06347386897033343|0.12658020959500457|2016|0.11589860584505572| 0.06347386897033343|0.12658020959500457|
|35211|  195|  Alabama Power Co|   AL|0.11589860584505572|0.06347386897033343|0.12658020959500457|2016|0.11589860584505572| 0.06347386897033343|0.12658020959500457|
|35212|  195|  Alabama Power Co|   AL|0.11589860584505572|0.06347386897033343|0.12658020959500457|2016|0.11589860584505572| 0.06347386897033343|0.12658020959500457|
|35213|  195|  Alabama Power Co|   AL|0.11589860584505572|0.06347386897033343|0.12658020959500457|2016|0.11589860584505572| 0.06347386897033343|0.12658020959500457|
|35058|  195|  Alabama Power Co|   AL|0.11589860584505572|0.06347386897033343|0.12658020959500457|2016|0.11589860584505572| 0.06347386897033343|0.12658020959500457|
|35054|  195|  Alabama Power Co|   AL|0.11589860584505572|0.06347386897033343|0.12658020959500457|2016|0.11589860584505572| 0.06347386897033343|0.12658020959500457|
|35055|  195|  Alabama Power Co|   AL|0.11589860584505572|0.06347386897033343|0.12658020959500457|2016|0.11589860584505572| 0.06347386897033343|0.12658020959500457|
|35052|  195|  Alabama Power Co|   AL|0.11589860584505572|0.06347386897033343|0.12658020959500457|2016|0.11589860584505572| 0.06347386897033343|0.12658020959500457|
|35053|  195|  Alabama Power Co|   AL|0.11589860584505572|0.06347386897033343|0.12658020959500457|2016|0.11589860584505572| 0.06347386897033343|0.12658020959500457|
|35051|  195|  Alabama Power Co|   AL|0.11589860584505572|0.06347386897033343|0.12658020959500457|2016|0.11589860584505572| 0.06347386897033343|0.12658020959500457|
|35584|  195|  Alabama Power Co|   AL|0.11589860584505572|0.06347386897033343|0.12658020959500457|2016|0.11589860584505572| 0.06347386897033343|0.12658020959500457|
|35585|  195|  Alabama Power Co|   AL|0.11589860584505572|0.06347386897033343|0.12658020959500457|2016|0.11589860584505572| 0.06347386897033343|0.12658020959500457|
+-----+-----+-----------------+-----+-------------------+-------------------+-------------------+----+-------------------+-------------------+-------------------+
only showing top 20 rows


scala>

scala> imputedData.coalesce(1)
res10: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [zip: string, eiaid: string ... 9 more fields]

scala>   .write
res11: org.apache.spark.sql.DataFrameWriter[org.apache.spark.sql.Row] = org.apache.spark.sql.DataFrameWriter@1c1f6e27

scala>   .option("header", "true")
res12: org.apache.spark.sql.DataFrameWriter[org.apache.spark.sql.Row] = org.apache.spark.sql.DataFrameWriter@1c1f6e27

scala>   .mode("overwrite")
res13: org.apache.spark.sql.DataFrameWriter[org.apache.spark.sql.Row] = org.apache.spark.sql.DataFrameWriter@1c1f6e27

scala>   .csv("/user/ar7165_nyu_edu/FinalProject/imputed_data")
```

The Profiling script performs data profiling by printing the schema, counting rows, and checking for null or NaN values in each column. To address missing data, it employs an Imputer transformer from Spark MLlib, targeting specific columns ('comm_rate', 'ind_rate', 'res_rate') to replace missing values with the mean of each column. The imputed DataFrame is then shown, and finally, the processed data is saved as a CSV file to a specified path.

Analytics

SSH-in-browser UPLOAD FILE DOWNLOAD FILE

```
scala> import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.SparkSession

scala> import org.apache.spark.sql.functions._
import org.apache.spark.sql.functions._

scala> import org.apache.spark.sql.expressions.Window
import org.apache.spark.sql.expressions.Window

scala>

scala> val spark = SparkSession.builder.appName("Electricity Rates Analysis").getOrCreate()
23/12/03 22:27:01 INFO org.apache.spark.SparkEnv: Registering MapOutputTracker
23/12/03 22:27:01 INFO org.apache.spark.SparkEnv: Registering BlockManagerMaster
23/12/03 22:27:01 INFO org.apache.spark.SparkEnv: Registering BlockManagerMasterHeartbeat
23/12/03 22:27:01 INFO org.apache.spark.SparkEnv: Registering OutputCommitCoordinator
spark: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@417d5b67

scala> import spark.implicits._
import spark.implicits._

scala>

scala> // 1. Read and Prepare Data

scala> val dataPath = "FinalProject/aggregated_electricity_rates_by_region/part-00000-29dfebc4-2a82-467f-b3bc-d0e4be1dc2cd-c000.csv"
dataPath: String = FinalProject/aggregated_electricity_rates_by_region/part-00000-29dfebc4-2a82-467f-b3bc-d0e4be1dc2cd-c000.csv

scala> val df = spark.read.option("header", "true").csv(dataPath)
df: org.apache.spark.sql.DataFrame = [region: string, year: string ... 5 more fields]

scala>

scala> // 2. Basic Data Exploration

scala> df.printSchema()
root
 |-- region: string (nullable = true)
 |-- year: string (nullable = true)
 |-- avg(comm_rate): string (nullable = true)
 |-- avg(ind_rate): string (nullable = true)
 |-- avg(res_rate): string (nullable = true)
 |-- avg(avg_rate): string (nullable = true)
```

SSH-in-browser UPLOAD FILE DOWNLOAD FILE

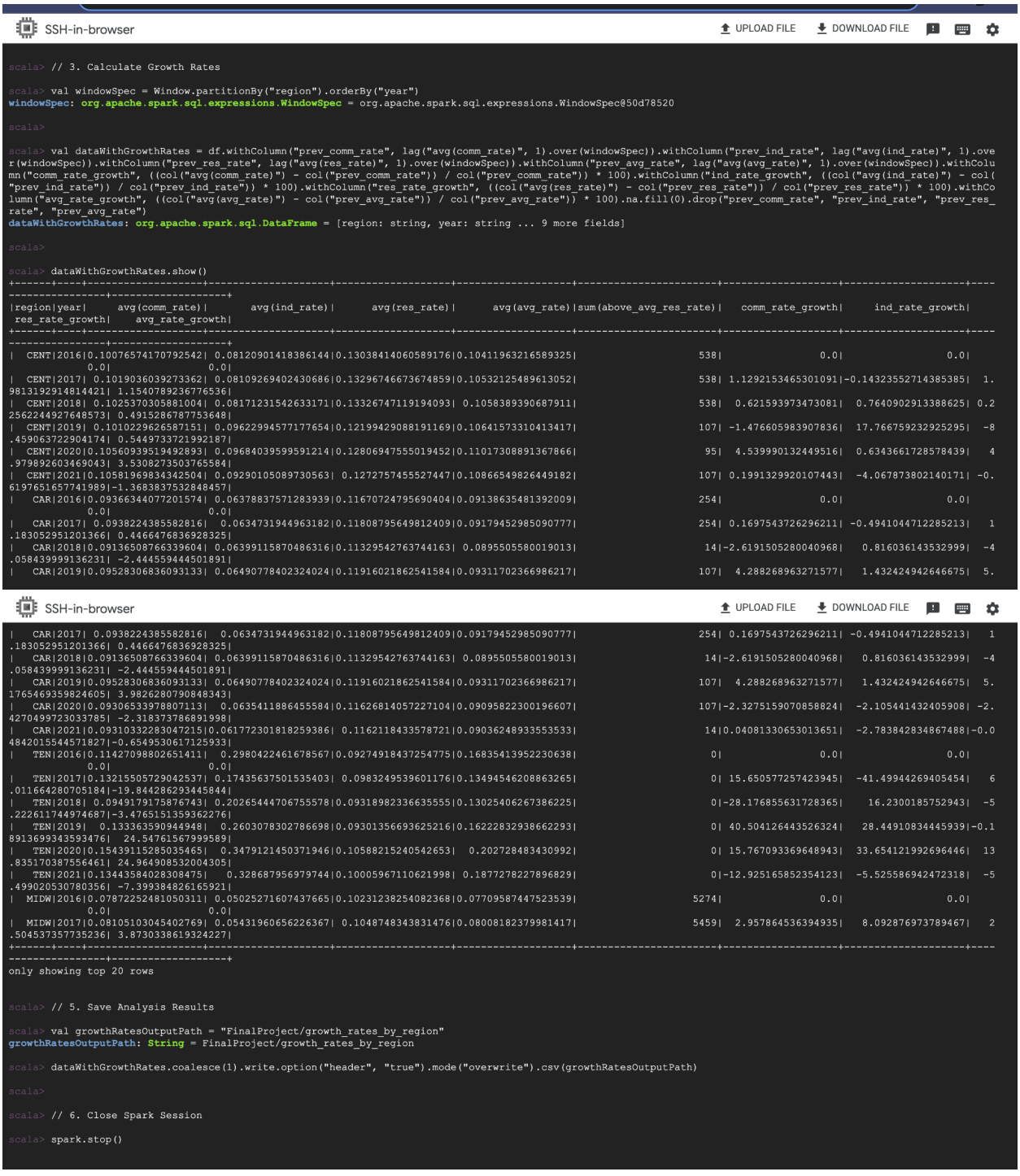
```
scala> // 2. Basic Data Exploration

scala> df.printSchema()
root
 |-- region: string (nullable = true)
 |-- year: string (nullable = true)
 |-- avg(comm_rate): string (nullable = true)
 |-- avg(ind_rate): string (nullable = true)
 |-- avg(res_rate): string (nullable = true)
 |-- avg(avg_rate): string (nullable = true)
 |-- sum(above_avg_res_rate): string (nullable = true)


scala> df.show()
+------+----+------------------+--------------------+-------------------+-------------------+-----------------------+
|region|year|     avg(comm_rate)|       avg(ind_rate)|      avg(res_rate)|      avg(avg_rate)|sum(above_avg_res_rate)|
+------+----+------------------+--------------------+-------------------+-------------------+-----------------------+
|  CENT|2017| 0.1019036039273362| 0.08109269402430686|0.13296746673674859|0.10532125489613052|                    538|
|    SW|2018| 0.0939536693329262| 0.06633169120851487|0.11315860656656936|0.09114798903600267|                    686|
|  MIDW|2018|0.08109520244049585|0.052688635066064284|0.10333875397309447|0.07904086382655084|                   5661|
|  MIDA|2019|0.08032935969012645| 0.04127802375016016| 0.0894596033013364|0.07035566224720739|                    835|
|   TEN|2016|0.11427098802651411|  0.2980422461678567|0.09274918437254775|0.16835413952230638|                      0|
|    NE|2017| 0.0856325986328245| 0.07049095674497964|0.11936085004249963| 0.09182813514010291|                   5150|
|    NY|2019|0.08029843598717201| 0.05357724671455165| 0.1215334296094146|0.08513637077037345|                   2105|
|  MIDW|2020|0.08305324585987275| 0.05256735302198046|0.10258668798652301|0.07940242895612597|                   5173|
|  CENT|2021|0.10581969834342504| 0.09290105089730563| 0.1272757455527447|0.10866549826449182|                    107|
|    SW|2017|0.09331554573430896| 0.06637878208729901|0.11462410795693762|0.09143947859284893|                    795|
|   TEN|2018| 0.0949179175876743| 0.20265444706755578|0.09318982336635555|0.13025406267386225|                      0|
|    SE|2020|0.08004188258746042| 0.05028728635874926|0.09506605232746813|0.07513174042455893|                   2264|
|    NE|2020|0.08450817566083985| 0.06794039274673558|0.12200352128739646|0.09148402989832623|                   5794|
|   TEN|2020|0.15439115285035465|  0.3479121450371946|0.10588215240542653|  0.202728483430992|                      0|
|    NE|2016|0.08458871460873893| 0.06686968401108087|0.11346102710686891|0.08830647524222936|                   5147|
|   TEX|2017|0.06635512371386913|0.043645879074515864|0.08656153247282122|0.06552084508706858|                    148|
|   CAR|2017| 0.0938224385582816|  0.0634731944963182|0.11808795649812409|0.09179452985090777|                    254|
|    NW|2018|0.07193323710821695|0.053221733974112916|0.07787572956103006| 0.0676769002144525|                    195|
|  MIDW|2016|0.07872252481050311| 0.05025271607437665|0.10231238254082368|0.07709587447523539|                   5274|
|    SW|2016| 0.0891281151765741| 0.06368201153218735|0.11075069602031772|0.08785360757635854|                    576|
+------+----+------------------+--------------------+-------------------+-------------------+-----------------------+
only showing top 20 rows


scala>

scala> // 3. Yearly and Regional Analysis
```

SSH-in-browser UPLOAD FILE DOWNLOAD FILE

```
scala> // 3. Yearly and Regional Analysis

scala> val yearlyRates = df.groupBy("year").agg(avg("avg(comm_rate)"), avg("avg(ind_rate)"), avg("avg(res_rate)"), avg("avg(avg_rate)"))
yearlyRates: org.apache.spark.sql.DataFrame = [year: string, avg(avg(comm_rate)): double ... 3 more fields]

scala> yearlyRates.show()
+----+------------------+------------------+------------------+------------------+
|year|avg(avg(comm_rate))| avg(avg(ind_rate))| avg(avg(res_rate))| avg(avg(avg_rate))|
+----+------------------+------------------+------------------+------------------+
|2016| 0.0894359810880749| 0.0793780621899012|0.10797553662766363| 0.0922631933018784|
|2020|0.09678271833719759|0.08578161416884099|0.11281733060900753|0.09846055437168322|
|2019|0.09420585613227811|0.07926960097500972|0.11105237701836343|0.09484261137521648|
|2017|0.09220914126332755|0.07146099232985126|0.11030502052841241|  0.091325051373864|
|2018|0.09046067171792276| 0.0756177118565198|0.11104656627163696|0.09237498328202641|
|2021| 0.1011652418651279|0.08901402757780846|0.11996172692292542|0.10338033212195362|
+----+------------------+------------------+------------------+------------------+


scala> val regionalRates = df.groupBy("region").agg(avg("avg(comm_rate)"), avg("avg(ind_rate)"), avg("avg(res_rate)"), avg("avg(avg_rate)"))
regionalRates: org.apache.spark.sql.DataFrame = [region: string, avg(avg(comm_rate)): double ... 3 more fields]

scala> regionalRates.show()
+------+------------------+------------------+------------------+------------------+
|region|avg(avg(comm_rate))|  avg(avg(ind_rate))| avg(avg(res_rate))| avg(avg(avg_rate))|
+------+------------------+------------------+------------------+------------------+
|  CENT|0.10294307207007185|  0.0883309027165824| 0.1289930984199054|0.10675569106885326|
|   CAR|  0.093383782995528| 0.06357900056684647|0.11662180577467146| 0.0911948631123488|
|   TEN| 0.1272557578305002| 0.26866016675772914|0.09720322535781993|0.16437304998201652|
|    SW|0.09295725390775557| 0.06530220964809846|0.11297849516169702| 0.0904126529058501|
|  MIDW|0.08202118420400986|0.053443771785435475|0.10432827577201825|0.07993107725382129|
|    NW|0.07326313728707996| 0.05309264532787348|0.07842547680989724|0.06826041980828358|
|    NE|0.08601392706754468|  0.0697288162009782|0.12010109959899296| 0.0919479476225054|
|   CAL|0.14943819956974277| 0.10772915407768868|0.17353460921982777|0.14356732095574945|
|    SE|0.07969560671848795|0.051883432095572585|0.09392177939398183|0.07516693940268164|
|    NY|0.08542298953369121|0.058558395293223424|0.12375927959610229| 0.0892468881410048|
|   FLA|0.09040632350077782| 0.0677214946435067 8|0.11245036538554772|0.09019272784327775|
|   TEX|0.07925150873014315| 0.05037941782165722|0.10557728499285608|0.07840273718155237|
|  MIDA|0.08050974579317956| 0.0427216127769924 8|0.09061541346903572|0.07128225734640305|
+------+------------------+------------------+------------------+------------------+


scala> // 4. Statistical Analysis

scala> val stats = df.describe("avg(comm_rate)", "avg(ind_rate)", "avg(res_rate)", "avg(avg_rate)")
```

SSH-in-browser UPLOAD FILE DOWNLOAD FILE

```
scala> // 4. Statistical Analysis

scala> val stats = df.describe("avg(comm_rate)", "avg(ind_rate)", "avg(res_rate)", "avg(avg_rate)")
stats: org.apache.spark.sql.DataFrame = [summary: string, avg(comm_rate): string ... 3 more fields]

scala> stats.show()
+-------+------------------+------------------+------------------+------------------+
|summary|     avg(comm_rate)|      avg(ind_rate)|      avg(res_rate)|      avg(avg_rate)|
+-------+------------------+------------------+------------------+------------------+
|  count|                78|                78|                78|                78|
|   mean|0.0940432684006547 9|0.08008700151632192| 0.1121930929963349 4| 0.0954411209711036 9|
| stddev|0.02231335210386406|0.06016805789585241|0.023700329376799688|0.028845776309599225|
|    min|0.06635512371386913| 0.0383057292913966| 0.0766675918024717 2| 0.0655208450870685 8|
|    max|0.17919488083346752| 0.3479121450371946| 0.2141833212652059 2|  0.202728483430992|
+-------+------------------+------------------+------------------+------------------+


scala> // 5. Regions with Most Above-Average Residential Rates

scala> val mostAboveAvgResRates = df.groupBy("region").agg(sum("sum(above_avg_res_rate)").alias("total_above_avg_res_rate")).orderBy(desc("total_above_avg_res_rate
"))
mostAboveAvgResRates: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [region: string, total_above_avg_res_rate: double]

scala> mostAboveAvgResRates.show()
+------+-----------------------+
|region|total_above_avg_res_rate|
+------+-----------------------+
|  MIDW|                32864.0|
|    NE|                32862.0|
|   CAL|                21832.0|
|    NY|                14385.0|
|    SE|                11908.0|
|  MIDA|                 4446.0|
|    SW|                 4223.0|
|  CENT|                 1923.0|
|    NW|                 1583.0|
|   TEX|                  887.0|
|   FLA|                  762.0|
|   CAR|                  750.0|
|   TEN|                    0.0|
+------+-----------------------+


scala> // 3. Calculate Growth Rates
```



In my Analytics Part, I did Yearly and Regional Analysis where I grouped the data by year and region and calculated the average of commercial, industrial, residential, and overall average rates for each year. For Statistical Analysis I used the describe function to provide summary statistics (like count, mean, standard deviation, min, and max) for the average rates of different types (commercial, industrial, residential, and overall average). I calculated the growth rate by

comparing the current year's rate with the previous year's rate for each category (commercial, industrial, residential, and overall average rate) within each region.