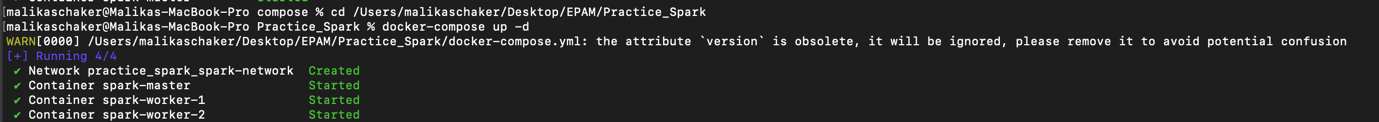
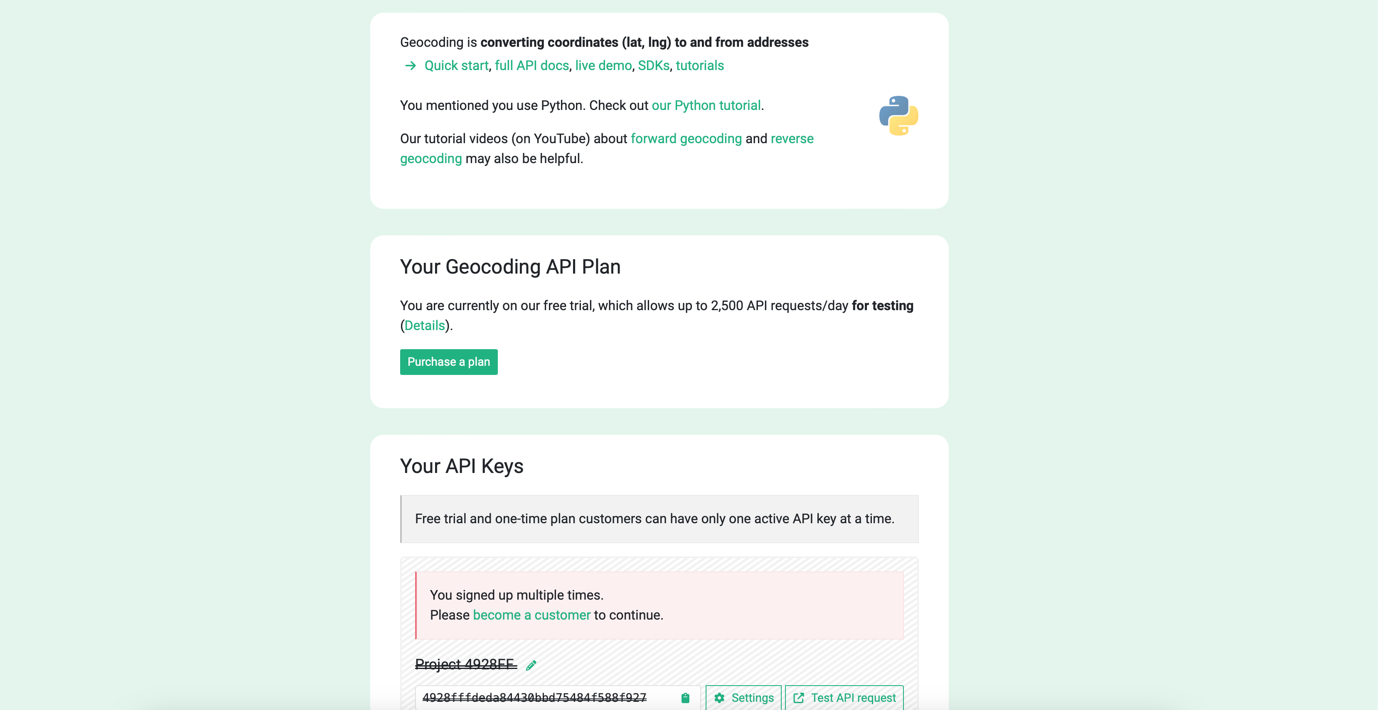
Install Spark locally using one of the methods described here or in Docker.



**STEP 1**

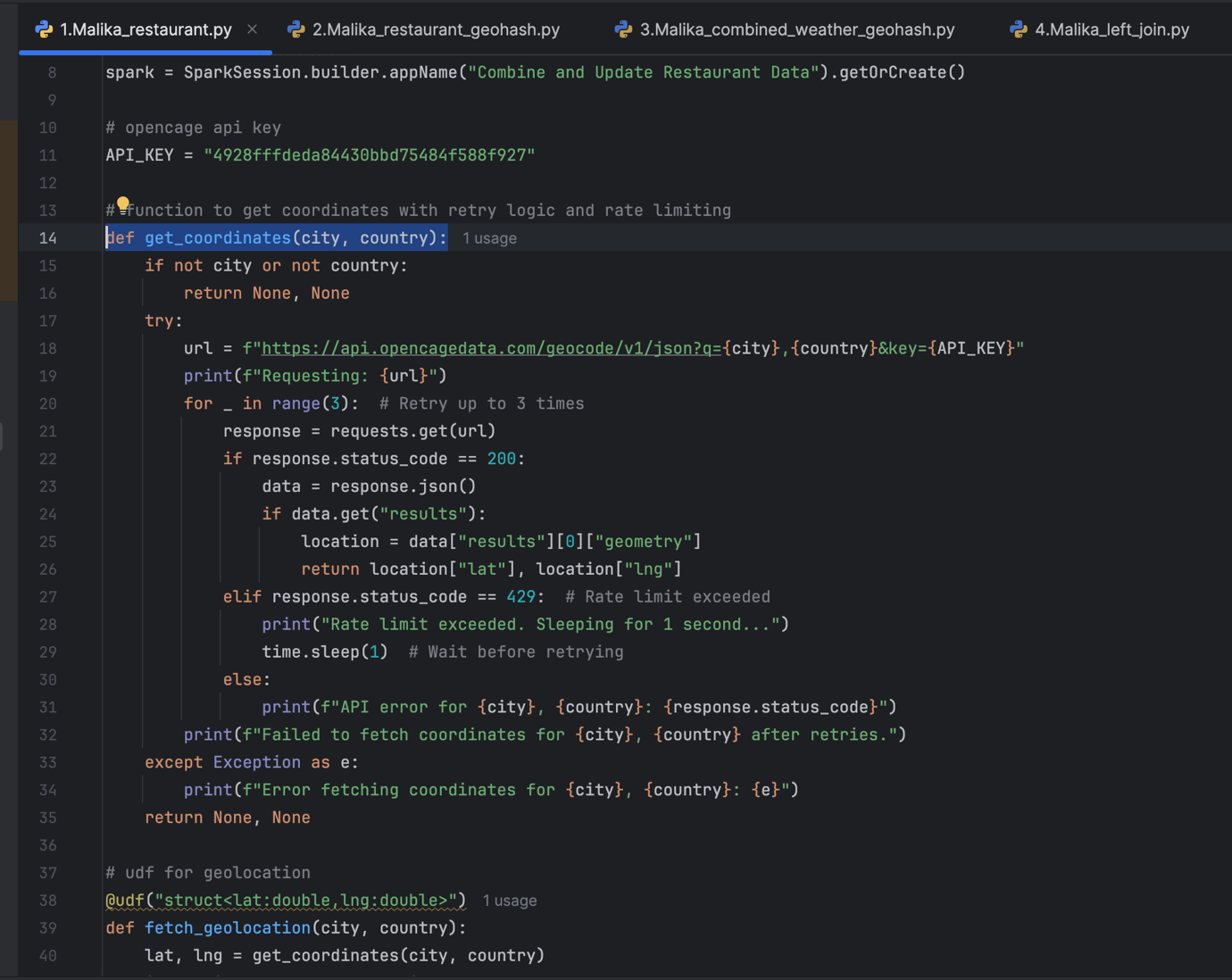
Sets up the environment for processing data using PySpark. A SparkSession is required to read, transform, and write large datasets efficiently.

The API key is used to authenticate requests to the OpenCage Geocoding API. Without a valid API key, requests to fetch latitude and longitude will fail.



Implements the function get\_coordinates(city, country), which:

* Sends a request to the OpenCage API for geocoding.
* Fetches latitude and longitude for the specified city and country.
* Handles errors and returns None for missing or invalid values.

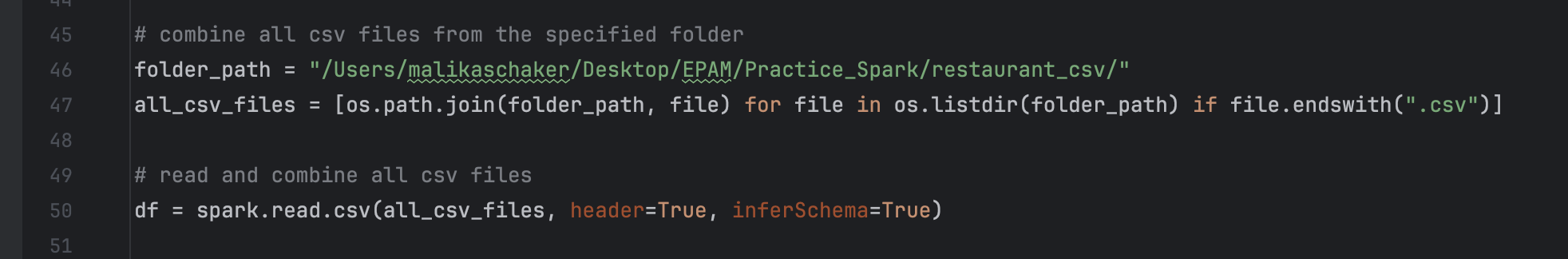


Registers a PySpark User Defined Function (UDF) called fetch\_geolocation. This UDF:

* Uses get\_coordinates to retrieve latitude and longitude.
* Returns a structured object containing the updated lat and lng values.
* Enables Spark to apply this function to each row in the dataset.

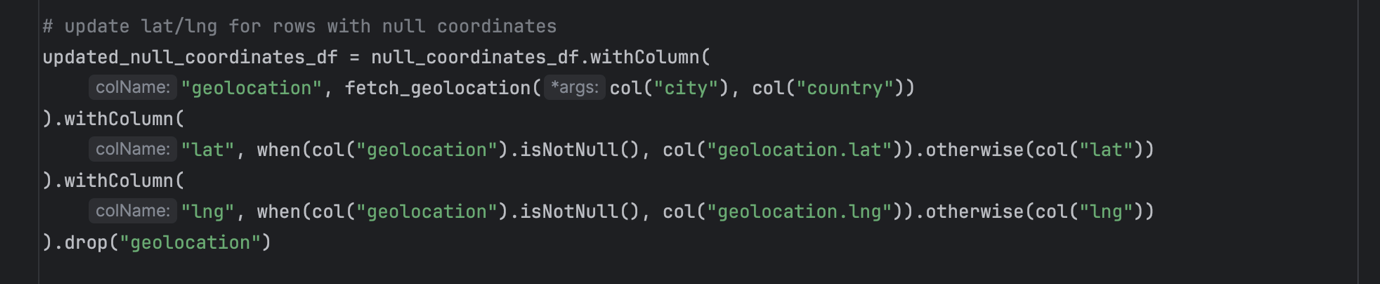
Loads the restaurant data from a CSV file into a Spark DataFrame.

* Ensures the data types are inferred (inferSchema=True) and the header row is used (header=True).
* Displays the original dataset to verify the structure and identify rows with missing latitude or longitude.



Applies the fetch\_geolocation UDF to update rows where latitude (lat) or longitude (lng) is missing.

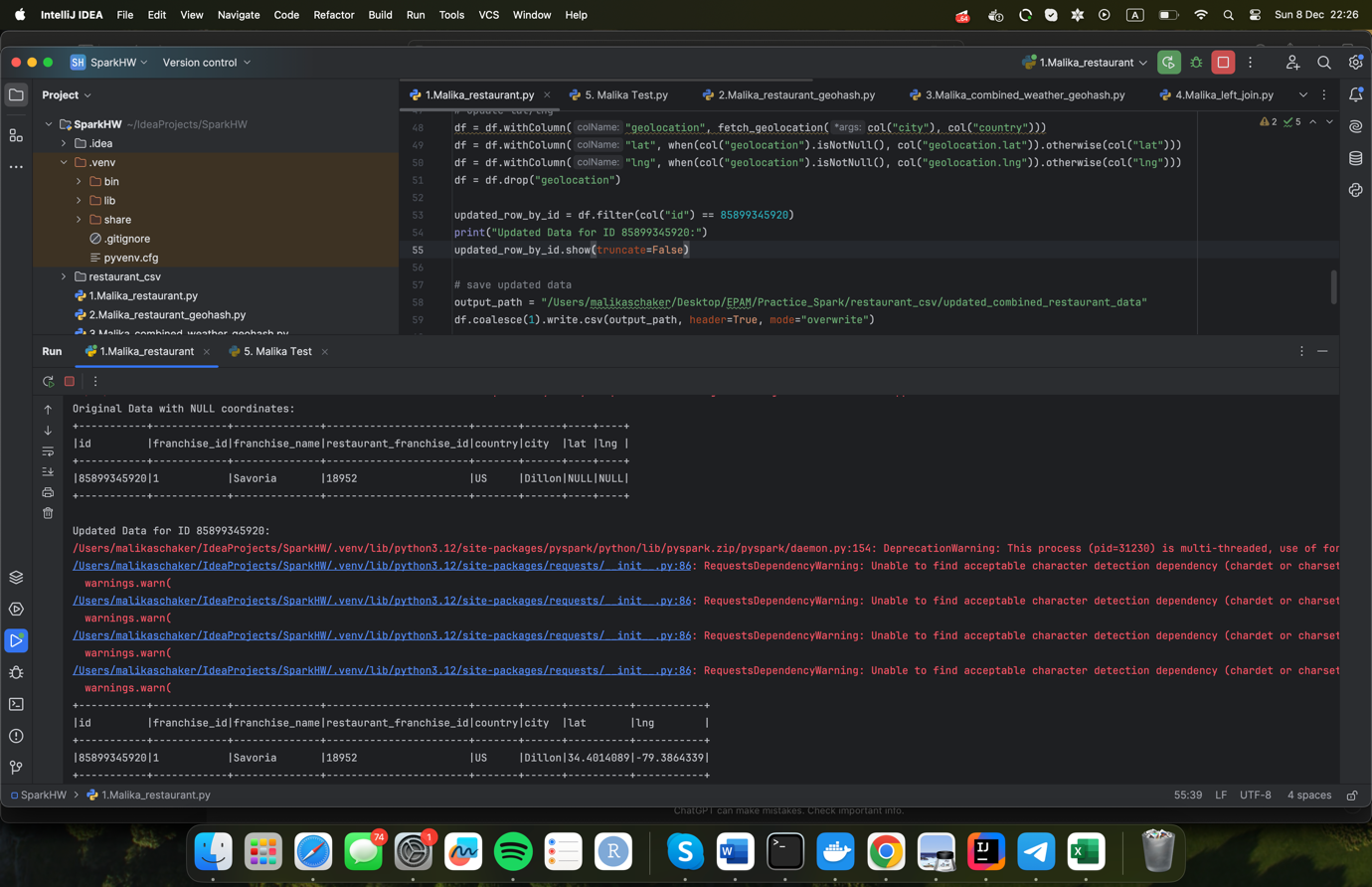
* For rows with valid geolocation data, lat and lng are updated.
* For rows without geolocation data, the original values remain unchanged.



Then it writes the updated DataFrame back to a CSV file.

* The file is saved with a single partition (coalesce(1)) for easier access.
* Overwrites any existing file in the specified output path.

Result:



**STEP 2**

spark = SparkSession.builder.appName("Generate Geohash").getOrCreate()

* Creates a SparkSession with the application name "Generate Geohash".
* A SparkSession is required for processing large-scale data with PySpark.

def generate\_geohash(lat, lng):

if lat is not None and lng is not None:

return pgh.encode(lat, lng, precision=4)

return None

* This function takes lat (latitude) and lng (longitude) as inputs.
* If both lat and lng are not None, it uses the pygeohash library to create a geohash with a precision of 4 characters.
* Returns None if either lat or lng is missing.

geohash\_udf = udf(generate\_geohash, StringType())

* Converts the generate\_geohash Python function into a PySpark UDF.
* This UDF can now be applied to a Spark DataFrame, processing each row to generate a geohash.

file\_path = "/Users/malikaschaker/Desktop/EPAM/Practice\_Spark/restaurant\_csv/updated\_combined\_restaurant\_data/part-00000-b01c3e06-773c-4602-98ef-b28fe99081ef-c000.csv"

df = spark.read.csv(file\_path, header=True, inferSchema=True)

* Reads the restaurant data from the specified CSV file into a Spark DataFrame.
* header=True: Indicates that the first row of the CSV contains column headers.
* inferSchema=True: Automatically infers the data types of the columns.

df\_with\_geohash = df.withColumn("geohash", geohash\_udf(df["lat"], df["lng"]))

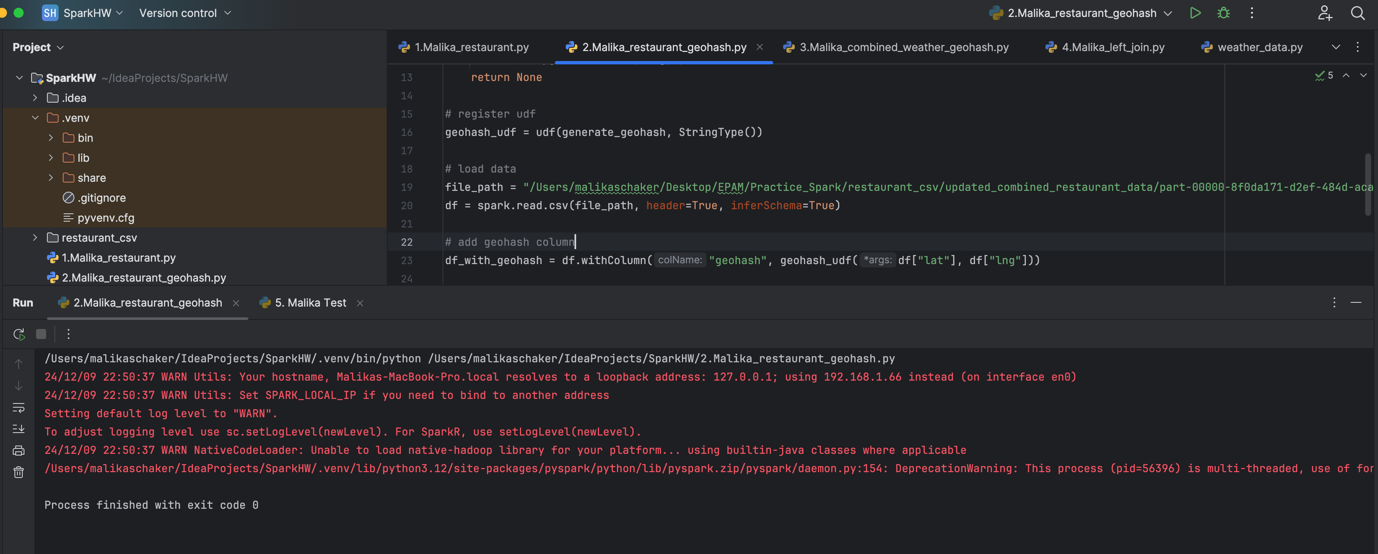
* Adds a new column, geohash, to the DataFrame.
* Uses the geohash\_udf to calculate the geohash for each row based on the lat and lng columns.

output\_path = "/Users/malikaschaker/Desktop/EPAM/Practice\_Spark/restaurant\_parquet/updated\_combined\_restaurant\_data\_with\_geohash"

df\_with\_geohash.write.parquet(output\_path, mode="overwrite")

* Saves the updated DataFrame (with the geohash column) as a .parquet file.
* mode="overwrite" ensures that any existing data at the output path is replaced.

Result:



**STEP 3**

spark = SparkSession.builder \

.appName("Combine Weather Data and Generate Geohash") \

.getOrCreate()

* Initializes a SparkSession with the application name "Combine Weather Data and Generate Geohash".
* This is required to process large-scale data efficiently using PySpark.

def generate\_geohash(lat, lng):

if lat is not None and lng is not None:

return pgh.encode(lat, lng, precision=4)

return None

* A Python function to generate a geohash for given lat (latitude) and lng (longitude).
* Uses the pygeohash library to encode coordinates with 4-character precision.
* Returns None if either lat or lng is missing.

geohash\_udf = udf(generate\_geohash, StringType())

* Registers the generate\_geohash function as a Spark UDF.
* This enables Spark to apply the function row by row in a DataFrame.

def find\_parquet\_files(base\_path):

all\_parquet\_files = []

for root, dirs, files in os.walk(base\_path):

for file in files:

if file.endswith(".parquet"):

all\_parquet\_files.append(os.path.join(root, file))

return all\_parquet\_files

* Recursively searches the base\_path directory for .parquet files.
* Returns a list of all .parquet file paths.

all\_parquet\_files = find\_parquet\_files(base\_path)

if not all\_parquet\_files:

print("No parquet files found. Exiting.")

exit()

* Collects all .parquet file paths from the specified base directory.
* If no files are found, prints a message and exits the script.

weather\_df = spark.read.parquet(\*all\_parquet\_files)

print(f"Loaded {weather\_df.count()} records from parquet files.")

* Reads and combines all .parquet files into a single Spark DataFrame.
* Uses \*all\_parquet\_files to pass the list of file paths as arguments.
* Prints the total number of records loaded.

weather\_df\_with\_geohash = weather\_df.withColumn("geohash", geohash\_udf(col("lat"), col("lng")))

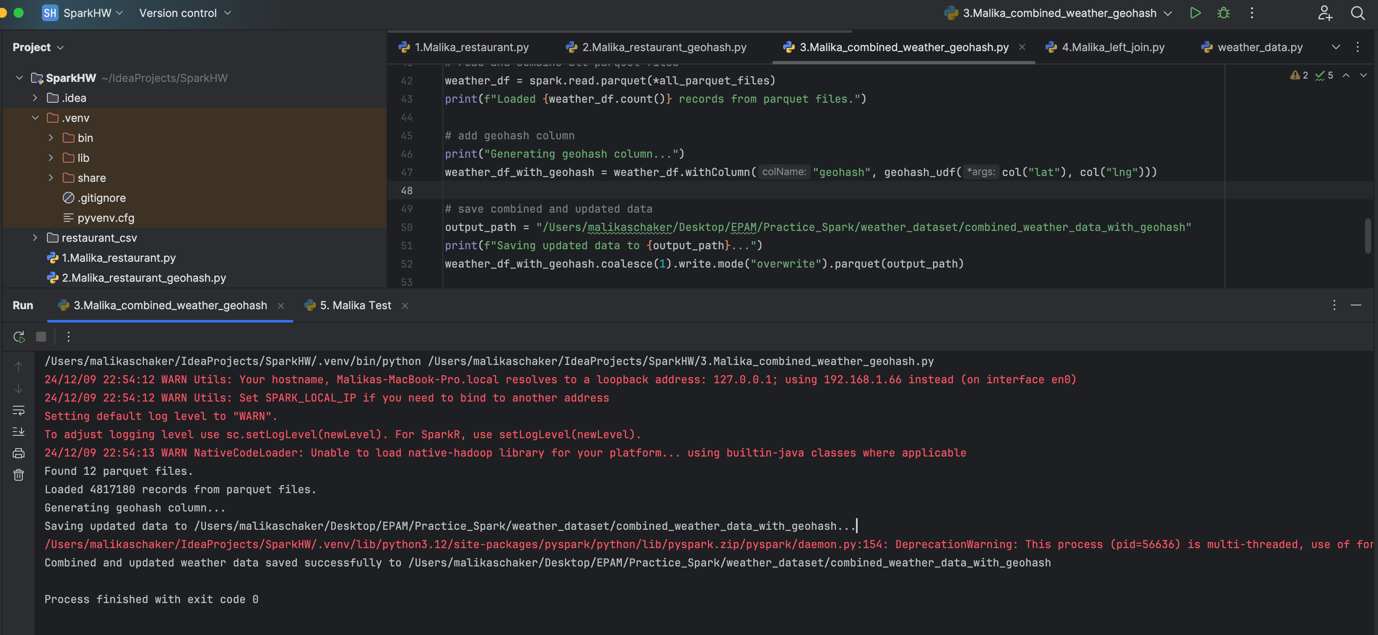
* Adds a new column geohash to the DataFrame.
* Uses the geohash\_udf to calculate the geohash for each row based on lat and lng.

output\_path = "/Users/malikaschaker/Desktop/EPAM/Practice\_Spark/weather\_dataset/combined\_weather\_data\_with\_geohash"

weather\_df\_with\_geohash.coalesce(1).write.mode("overwrite").parquet(output\_path)

* Saves the updated DataFrame with the geohash column to a single .parquet file.
* coalesce(1) ensures the output is written as a single file.
* mode("overwrite") overwrites any existing data at the specified path.

Result:



**STEP 4**

spark = SparkSession.builder.appName("Left Join Weather and Restaurant Data").getOrCreate()

* Creates a SparkSession named "Left Join Weather and Restaurant Data".
* This session is required to load, process, and join datasets using PySpark.

weather\_data\_path = "/Users/malikaschaker/Desktop/EPAM/Practice\_Spark/weather\_dataset/combined\_weather\_data\_with\_geohash"

restaurant\_data\_path = "/Users/malikaschaker/Desktop/EPAM/Practice\_Spark/restaurant\_parquet/updated\_combined\_restaurant\_data\_with\_geohash"

* Specifies the file paths for the weather and restaurant datasets.
* Both datasets are stored in .parquet format and include a geohash column.

weather\_df = spark.read.parquet(weather\_data\_path)

restaurant\_df = spark.read.parquet(restaurant\_data\_path)

* Loads the weather and restaurant datasets into separate Spark DataFrames.
* The .parquet format allows for efficient storage and quick loading of large datasets.

joined\_df = restaurant\_df.join(weather\_df, on="geohash", how="left")

* Performs a left join on the two datasets using the geohash column as the key.
* The resulting DataFrame (joined\_df) contains all rows from the restaurant\_df and matching rows from the weather\_df.
* If no match is found in weather\_df, the corresponding fields in joined\_df will have NULL values.

joined\_df.show(n=200, truncate=False)

* Displays the first 200 rows of the joined DataFrame in a non-truncated format.
* Useful for verifying the results and ensuring the join was performed correctly.

output\_path = "/Users/malikaschaker/Desktop/EPAM/Practice\_Spark/enriched\_weather\_restaurant\_data"

joined\_df.write.partitionBy("geohash").mode("overwrite").parquet(output\_path)

* Saves the joined dataset to the specified output\_path in .parquet format.
* Data is partitioned by the geohash column, which creates subdirectories for each unique geohash.
* The mode("overwrite") ensures that any existing data in the output directory is replaced.

Result:

