Project

February 5, 2024

[ ]:
```python
# import pyspark, config Mongo and initiate
import pyspark
from pyspark.sql import SparkSession

# MONGO CONFIGURATION
mongo_uri = "mongodb://admin:mongopw@mongo:27017/demo.feedback?authSource=admin"

# Spark init
spark = SparkSession \
    .builder \
    .master("local") \
    .appName('jupyter-pyspark') \
      .config("spark.mongodb.input.uri", mongo_uri) \
      .config("spark.mongodb.output.uri", mongo_uri) \
      .config("spark.jars.packages","org.mongodb.spark:mongo-spark-connector_2.
↪12:3.0.1")\
    .getOrCreate()
sc = spark.sparkContext
sc.setLogLevel("ERROR")

print(mongo_uri)
```

[ ]:
```python
# Create a Spark session
spark = SparkSession.builder.appName("CSV to MongoDB").getOrCreate()

# Source CSV file path
source_file = "file:///home/jovyan//Project//WeatherEvents_Jan2016-Dec2022.csv"
# Read CSV file into a DataFrame
weather_data = spark.read.csv(source_file, header=True, inferSchema=True)

# Filter data for the state of South Carolina (SC)
sc_weather_data = weather_data.filter(weather_data['State'] == 'SC')

# Write DataFrame to MongoDB collection
sc_weather_data.write.format("mongo") \
    .mode("overwrite") \
    .option("database", "project") \
```

```python
    .option("collection", "projectweather") \
    .save()
print(source_file)
```

[ ]:
```python
# removing columns that are unneeded
# Create a Spark session
spark = SparkSession.builder.appName("Remove Columns").getOrCreate()

columns_to_drop = ['locationlat', 'locationlng', 'AirportCode']
sc_weather_data = sc_weather_data.drop(*columns_to_drop)

sc_weather_data.show()
```

[ ]:
```python
from pyspark.sql.functions import col

# Drop null values from specific columns
columns_to_drop_null = ['Type']
sc_weather_data = sc_weather_data.dropna(subset=columns_to_drop_null)

# Show the cleaned DataFrame
sc_weather_data.show()
```

[ ]:
```python
from pyspark.sql.functions import col

# Select the columns of interest
selected_columns = ['Type', 'Precipitation(in)', 'ZipCode']

# Calculate summary statistics for the selected columns
summary_stats = sc_weather_data.select(selected_columns).summary()

# Show the summary statistics
summary_stats.show()
```

[ ]:
```python
import matplotlib.pyplot as plt

# Grouping by 'Type' and counting occurrences
type_counts = sc_weather_data.groupBy('Type').count().orderBy('Type')

# Convert PySpark DataFrame to Pandas for plotting
type_counts_pandas = type_counts.toPandas()

# Plotting the line chart
plt.figure(figsize=(10, 6))
plt.plot(type_counts_pandas['Type'], type_counts_pandas['count'], marker='o')
plt.xlabel('Type')
plt.ylabel('Count')
plt.title('Count of each Type in sc_weather_data')
```

```
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability if needed
plt.grid(True)
plt.tight_layout()

# Show the plot
plt.show()
```

[ ]:
```
import matplotlib.pyplot as plt

# Grouping by 'Type' and counting occurrences
type_counts = sc_weather_data.groupBy('Severity').count().orderBy('Severity')

# Convert PySpark DataFrame to Pandas for plotting
type_counts_pandas = type_counts.toPandas()

# Plotting the line chart
plt.figure(figsize=(10, 6))
plt.plot(type_counts_pandas['Severity'], type_counts_pandas['count'],␣
↪marker='o')
plt.xlabel('Severity')
plt.ylabel('Count')
plt.title('Count of each Severity in sc_weather_data')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability if needed
plt.grid(True)
plt.tight_layout()

# Show the plot
plt.show()
```

[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from pyspark.sql import SparkSession

# Assuming you have a SparkSession named 'spark'
spark = SparkSession.builder.appName("BoxPlot").getOrCreate()

# Assuming 'sc_weather_data' is your PySpark DataFrame

# Collect precipitation data for each ZipCode
grouped_data = sc_weather_data.groupBy('ZipCode').agg({'Precipitation(in)':␣
↪'collect_list'}).collect()

# Prepare data for box plot
data = [row['collect_list(Precipitation(in))'] for row in grouped_data]

# Create a box plot
plt.figure(figsize=(10, 6))
```

```
sns.boxplot(data=data)
plt.xlabel('Zip Codes')
plt.ylabel('Precipitation (inches)')
plt.title('Distribution of Precipitation across Zip Codes')
plt.tight_layout()

# Show the plot
plt.show()
```

[ ]:
```
sc_weather_data.describe()
```

[ ]:
```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
from statsmodels.tsa.arima.model import ARIMA
import pandas as pd

# Create a Spark session (if not created already)
spark = SparkSession.builder.appName("ARIMA").getOrCreate()

# Convert 'StartTime(UTC)' column to timestamp type in PySpark DataFrame
sc_weather_data = sc_weather_data.withColumn("StartTime(UTC)",
 ↪col("StartTime(UTC)").cast("timestamp"))

# Register DataFrame as a temporary view
sc_weather_data.createOrReplaceTempView("weather_data")

# Modify the SQL query to correctly reference the 'StartTime(UTC)' and
 ↪'Precipitation(in)' columns
data_resampled = spark.sql("""
    SELECT DATE(`StartTime(UTC)`) AS Date, SUM(`Precipitation(in)`) AS
 ↪TotalPrecipitation
    FROM weather_data
    GROUP BY DATE(`StartTime(UTC)`)
    ORDER BY DATE(`StartTime(UTC)`)
""")

# Convert PySpark DataFrame to Pandas DataFrame for ARIMA modeling
data_resampled_pd = data_resampled.toPandas()

# Set the 'Date' column as the index
data_resampled_pd.set_index('Date', inplace=True)

# Convert 'TotalPrecipitation' column to numeric type in Pandas DataFrame
data_resampled_pd['TotalPrecipitation'] = pd.
 ↪to_numeric(data_resampled_pd['TotalPrecipitation'], errors='coerce')

# Fill NaN values with 0 or any other suitable method based on your data
```

```
data_resampled_pd.fillna(0, inplace=True)  # Fill NaN values with 0 for example

# Fit an ARIMA model using Pandas DataFrame
model = ARIMA(data_resampled_pd['TotalPrecipitation'], order=(1, 2, 1))
results = model.fit()

# Generate forecasts (example: forecasting 10 steps ahead)
forecast_steps = 210
forecast = results.forecast(steps=forecast_steps)

print(forecast)
```

[ ]:
```
from datetime import datetime, timedelta

start_date = datetime(2016, 1, 6)

# List of forecast indexes
forecast_indexes =␣
↪[2737,2738,2739,2740,2741,2742,2743,2744,2745,2746,2561,2562,2563,2564,2565,2566,2567,2568,

# Assuming daily frequency, calculate dates for each forecast index
for index in forecast_indexes:
    forecast_date = start_date + timedelta(days=index)
    print(f"Date for forecast step {index}: {forecast_date}")
```

[ ]:
```
import matplotlib.pyplot as plt
from datetime import datetime, timedelta

# Starting date of the data
start_date = datetime(2016, 1, 6)

# Provided forecast indexes and values
forecast_indexes =␣
↪[2551,2552,2553,2554,2555,2556,2557,2558,2559,2560,2561,2562,2563,2564,2565,2566,2567,2568,
forecast_values = [5.090654,5.091008,5.091521,5.091973,5.092448,5.092915,5.
↪093385,5.093853,5.094323,5.094791,5.09526,5.095729,5.096198,5.096667,5.
↪097136,5.097605,5.098074,5.098543,5.099012,5.099481,5.09995]
# Calculate dates for each forecast index starting from the provided start date
forecast_dates = [start_date + timedelta(days=index - 2551) for index in␣
↪forecast_indexes]

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(forecast_dates, forecast_values, marker='o', linestyle='-',␣
↪color='green')
plt.title('Forecasted Values over Time')
plt.xlabel('Date')
```

```
plt.ylabel('Forecasted Value')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.tight_layout()
plt.show()
```

[ ]:
```
import matplotlib.pyplot as plt
from datetime import datetime, timedelta

# Starting date of the data
start_date = datetime(2016, 1, 6)

# Provided forecast indexes and values
forecast_indexes = [2737,2738,2739,2740,2741,2747,2748,2749,2750,2751]
forecast_values = [5.177796,5.178265,5.178733,5.179202,5.179671,5.182485,5.
 ↪182954,5.183423,5.183892,5.184361]

# Calculate dates for each forecast index starting from the provided start date
forecast_dates = [start_date + timedelta(days=index - 2551) for index in␣
 ↪forecast_indexes]

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(forecast_dates, forecast_values, marker='o', linestyle='-',␣
 ↪color='green')
plt.title('Forecasted Values over Time')
plt.xlabel('Date')
plt.ylabel('Forecasted Value')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.tight_layout()
plt.show()
```

[ ]:
```
import matplotlib.pyplot as plt
from pyspark.sql import SparkSession

# Assuming you have a SparkSession named 'spark'
spark = SparkSession.builder.appName("TypePerZipCode").getOrCreate()

# Assuming 'sc_weather_data' is your PySpark DataFrame

# Grouping by 'ZipCode' and 'Type', counting occurrences, and ordering by count␣
 ↪in descending order
zipcode_counts = sc_weather_data.groupBy('ZipCode').count().orderBy('count',␣
 ↪ascending=False)

# Selecting top 10 ZipCodes
top_10_zipcodes = zipcode_counts.limit(10).select('ZipCode')
```

```python
# Filtering 'sc_weather_data' to keep only records with ZipCodes present in the␣
↪top 10 list
filtered_data = sc_weather_data.join(top_10_zipcodes, 'ZipCode', 'inner')

# Grouping by 'ZipCode' and 'Type', counting occurrences
type_per_zipcode = filtered_data.groupBy('ZipCode', 'Type').count().
↪orderBy('ZipCode', 'Type')

# Collecting the data to the driver as a Pandas DataFrame for plotting
type_per_zipcode_pandas = type_per_zipcode.toPandas()

# Plotting the bar graph
plt.figure(figsize=(12, 6))
type_per_zipcode_pandas.pivot(index='ZipCode', columns='Type', values='count').
↪plot(kind='bar', stacked=True)
plt.xlabel('ZipCode')
plt.ylabel('Count')
plt.title('Count of Types per Top 10 ZipCodes')
plt.legend(title='Type', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()

# Show the plot
plt.show()
```