Vrushali_new

December 17, 2024

[1]:
```
spark = SparkSession.builder.appName("NYPD").getOrCreate()
spark.sparkContext.setLogLevel("ERROR")
```

```
24/12/17 01:11:53 WARN SparkSession: Using an existing Spark session; only
runtime SQL configurations will take effect.
```

[2]:
```
import pyspark.sql.functions as F
import matplotlib.pyplot as plt
import pandas as pd
import folium
from folium.plugins import HeatMap
```

[3]:
```
#https://storage.cloud.google.com/nypdbucket/notebooks/jupyter/nypd_dataset.csv
csv_file_path = "gs://nypdbucket/notebooks/jupyter/nypd_dataset.csv"


# Read the CSV file into a DataFrame
df = (spark.read.format("csv")
     .option("header", "true")  # If the CSV has a header row
     .option("inferSchema", "true")  # Automatically infer the schema
     .load(csv_file_path))


# Show the first few rows of the DataFrame
df.show(5)
```

```
[Stage 2:>                                                          (0 + 1) / 1]

+-----------+-----------+-------------+-------------------+-----------+--------
+--------------------+--------+--------+----------+--------------------+--------
+--------------------+--------------------+--------------------+----------------
----+---------+----------+
|CAD_EVNT_ID|CREATE_DATE|INCIDENT_DATE|      INCIDENT_TIME|NYPD_PCT_CD| BORO_NM|
PATRL_BORO_NM|GEO_CD_X|GEO_CD_Y|RADIO_CODE|            TYP_DESC|CIP_JOBS|
ADD_TS|              DISP_TS|            ARRIVD_TS|           CLOSNG_TS| Latitude|
Longitude|
+-----------+-----------+-------------+-------------------+-----------+--------
+--------------------+--------+--------+----------+--------------------+--------
+--------------------+--------------------+--------------------+----------------
```

```
----+---------+----------+
|   99842231| 01/01/2024|   12/31/2023|2024-12-17 22:11:38|         45|   BRONX|
PATROL BORO BRONX| 1031438|  249344|      52D6|     DISPUTE: FAMILY| Non
CIP|01/01/2024 12:02:…|01/01/2024 12:02:…|               NULL|01/01/2024
12:13:…|40.850949|-73.829434|
|   99842388| 01/01/2024|   12/31/2023|2024-12-17 22:19:46|        110|
QUEENS|PATROL BORO QUEEN…| 1022087|  208229|      52D6|     DISPUTE: FAMILY|
Non CIP|01/01/2024 12:09:…|01/01/2024 12:11:…|
NULL|01/01/2024 12:57:…|40.738144|-73.863466|
|   99842587| 01/01/2024|   12/31/2023|2024-12-17 22:31:06|        108|
QUEENS|PATROL BORO QUEEN…| 1007298|  209993|       53S|VEHICLE ACCIDENT:…|
Non CIP|01/01/2024 12:01:…|01/01/2024 12:02:…|01/01/2024
01:19:…|01/01/2024 01:20:…|40.743037|-73.916826|
|   99843964| 01/01/2024|   12/31/2023|2024-12-17 23:53:22|        114|
QUEENS|PATROL BORO QUEEN…| 1002279|  222019|      34K1|ASSAULT (IN
PROGR…|Critical|01/01/2024 12:06:…|01/01/2024 12:07:…|01/01/2024
12:19:…|01/01/2024 01:03:…|40.776057|-73.934906|
|   99844026| 01/01/2024|   12/31/2023|2024-12-17 23:57:38|
66|BROOKLYN|PATROL BORO BKLYN…|  987908|  174328|      11C4|ALARMS:
COMMERCIA…| Non CIP|01/01/2024 12:04:…|01/01/2024 01:45:…|
NULL|01/01/2024 02:23:…|40.645174| -73.98682|
+-----------+----------+------------+-------------------+-----------+--------
+--------------------+--------+--------+----------+--------------------+--------
+--------------------+--------------------+--------------------+----------------
----+---------+----------+
only showing top 5 rows
```

[4]:
```python
# Print the schema of the DataFrame
df.printSchema()
```

```
root
 |-- CAD_EVNT_ID: integer (nullable = true)
 |-- CREATE_DATE: string (nullable = true)
 |-- INCIDENT_DATE: string (nullable = true)
 |-- INCIDENT_TIME: timestamp (nullable = true)
 |-- NYPD_PCT_CD: integer (nullable = true)
 |-- BORO_NM: string (nullable = true)
 |-- PATRL_BORO_NM: string (nullable = true)
 |-- GEO_CD_X: integer (nullable = true)
 |-- GEO_CD_Y: integer (nullable = true)
 |-- RADIO_CODE: string (nullable = true)
 |-- TYP_DESC: string (nullable = true)
 |-- CIP_JOBS: string (nullable = true)
 |-- ADD_TS: string (nullable = true)
 |-- DISP_TS: string (nullable = true)
 |-- ARRIVD_TS: string (nullable = true)
```

```
 |-- CLOSNG_TS: string (nullable = true)
 |-- Latitude: double (nullable = true)
 |-- Longitude: double (nullable = true)
```

[5]:
```
df.describe().show()
```

```
24/12/17 01:12:15 WARN SparkStringUtils: Truncated the string representation of
a plan since it was too large. This behavior can be adjusted by setting
'spark.sql.debug.maxToStringFields'.
[Stage 5:>                                                          (0 + 1) / 1]

+-------+-------------------+-----------+-------------+----------------+-------
------+--------------------+------------------+------------------+---------------
----+--------------------+--------+--------------------+--------------------+---
-----------------+--------------------+--------------------+-------------------+
|summary|        CAD_EVNT_ID|CREATE_DATE|INCIDENT_DATE|      NYPD_PCT_CD|
BORO_NM|       PATRL_BORO_NM|          GEO_CD_X|         GEO_CD_Y|
RADIO_CODE|             TYP_DESC|CIP_JOBS|              ADD_TS|
DISP_TS|           ARRIVD_TS|           CLOSNG_TS|           Latitude|
Longitude|
+-------+-------------------+-----------+-------------+----------------+-------
------+--------------------+------------------+------------------+---------------
----+--------------------+--------+--------------------+--------------------+---
-----------------+--------------------+--------------------+-------------------+
|  count|            5430525|    5430525|      5430525|          5430524|
5430525|             5430525|           5430525|          5430525|
5430525|             5430525| 5430525|             5430525|             5430525|
4292498|             5430492|             5430525|            5430525|
|   mean|1.031539814432809E8|       NULL|         NULL| 61.0436114452307|
NULL|                NULL|1003719.7164797143|207219.3408618872|
4.88391838826703E8|               NULL|    NULL|                NULL|
NULL|                NULL|                NULL|  40.73540911380572|
-73.92972289303498|
| stddev| 1915287.3223354751|       NULL|         NULL|34.59641421997529|
NULL|
NULL|20214.115777641575|29594.63603927419|5.062920631799014E9|
NULL|    NULL|                NULL|                NULL|                NULL|
NULL|0.08123306598960091|0.07290432332565411|
|    min|           99842231| 01/01/2024|   01/01/2024|                0|
(null)|              (null)|            913411|           121022|
001|10-53 NO RMP REQU…|Critical|01/01/2024 01:00:…|01/01/2024
01:00:…|01/01/2024 01:00:…|01/01/2024 01:00:…|          40.498596|
-74.254743|
|    max|          106487582| 09/30/2024|   12/31/2023|              123|STATEN
ISLAND|PATROL BORO STATE…|           1067305|           272307|
DEPTOW|    YOUTH HOME VISIT| Serious|09/30/2024 12:59:…|10/01/2024
12:53:…|10/01/2024 12:59:…|10/01/2024 12:59:…|          40.914065|
-73.700291|
```

```
+-------+-------------------+-----------+-------------+-----------------+-------
------+--------------------+------------------+-----------------+---------------
----+--------------------+--------+--------------------+--------------------+---
-----------------+--------------------+-------------------+-------------------+
```

1 Data Cleaning

[6]:
```python
print("Duplicate Rows:", df.count() - df.distinct().count())
```

```
[Stage 11:=====================================================>    (12 + 1) / 13]
```

```
Duplicate Rows: 0
```

[7]:
```python
arr_null_count = df.filter(F.col("ARRIVD_TS").isNull()).count()
clo_null_count = df.filter(F.col("CLOSNG_TS").isNull()).count()
pct_null_count = df.filter(F.col("NYPD_PCT_CD").isNull()).count()

print(f"Number of rows with NULL values in ARRIVD_TS: {arr_null_count}")
print(f"Number of rows with NULL values in CLOSNG_TS: {clo_null_count}")
print(f"Number of rows with NULL values in NYPD_PCT_CD: {pct_null_count}")
```

```
[Stage 21:=====================================>                    (8 + 4) / 12]
```

```
Number of rows with NULL values in ARRIVD_TS: 1138027
Number of rows with NULL values in CLOSNG_TS: 33
Number of rows with NULL values in NYPD_PCT_CD: 1
```

[8]:
```python
# Drop rows where ARRIVD_TS or CLOSNG_TS is NULL
df_cleaned = df.filter((F.col("ARRIVD_TS").isNotNull()) & (F.col("CLOSNG_TS").
 ↪isNotNull()))

# Show counts after dropping rows
print("Total rows after dropping NULL values in ARRIVD_TS and CLOSNG_TS:",␣
 ↪df_cleaned.count())

# Verify NULLs again
df_cleaned.select(
    F.col("ARRIVD_TS").isNull().alias("ARRIVD_TS_NULL"),
    F.col("CLOSNG_TS").isNull().alias("CLOSNG_TS_NULL")
).show(5)
```

```
Total rows after dropping NULL values in ARRIVD_TS and CLOSNG_TS: 4292472
+--------------+--------------+
|ARRIVD_TS_NULL|CLOSNG_TS_NULL|
+--------------+--------------+
|         false|         false|
|         false|         false|
|         false|         false|
|         false|         false|
|         false|         false|
+--------------+--------------+
only showing top 5 rows
```

[9]:
```python
# Fill NULL value in NYPD_PCT_CD with 'UNKNOWN'
df_cleaned = df_cleaned.fillna({"NYPD_PCT_CD": "UNKNOWN"})

# Verify NULLs in NYPD_PCT_CD
print("NULL values in NYPD_PCT_CD after filling:")
df_cleaned.filter(F.col("NYPD_PCT_CD").isNull()).show()
```

```
NULL values in NYPD_PCT_CD after filling:
[Stage 30:=======================>                                (3 + 4) / 7]
+-----------+-----------+-------------+-------------+-----------+-------+-------
------+--------+--------+----------+--------+--------+------+-------+---------+-
--------+--------+---------+
|CAD_EVNT_ID|CREATE_DATE|INCIDENT_DATE|INCIDENT_TIME|NYPD_PCT_CD|BORO_NM|PATRL_B
ORO_NM|GEO_CD_X|GEO_CD_Y|RADIO_CODE|TYP_DESC|CIP_JOBS|ADD_TS|DISP_TS|ARRIVD_TS|C
LOSNG_TS|Latitude|Longitude|
+-----------+-----------+-------------+-------------+-----------+-------+-------
------+--------+--------+----------+--------+--------+------+-------+---------+-
--------+--------+---------+
+-----------+-----------+-------------+-------------+-----------+-------+-------
------+--------+--------+----------+--------+--------+------+-------+---------+-
--------+--------+---------+


 [Stage 30:===============================================>        (6 + 1) /
7]
```

[10]:
```python
# Verify NULL counts for all relevant columns
df_cleaned.select(
    F.count(F.when(F.col("ARRIVD_TS").isNull(), 1)).
  ↪alias("ARRIVD_TS_NULL_COUNT"),
    F.count(F.when(F.col("CLOSNG_TS").isNull(), 1)).
  ↪alias("CLOSNG_TS_NULL_COUNT"),
    F.count(F.when(F.col("NYPD_PCT_CD").isNull(), 1)).
  ↪alias("NYPD_PCT_CD_NULL_COUNT")
```

```
).show()
```

```
[Stage 31:==================================================>    (11 + 1) / 12]

+--------------------+--------------------+----------------------+
|ARRIVD_TS_NULL_COUNT|CLOSNG_TS_NULL_COUNT|NYPD_PCT_CD_NULL_COUNT|
+--------------------+--------------------+----------------------+
|                   0|                   0|                     0|
+--------------------+--------------------+----------------------+
```

[11]:
```
# Drop unnecessary columns
df_cleaned = df_cleaned.drop("CAD_EVNT_ID", "PATRL_BORO_NM")

# Verify the columns after dropping
print("Columns after dropping CAD_EVNT_ID and PATRL_BORO_NM:")
df_cleaned.printSchema()
```

```
Columns after dropping CAD_EVNT_ID and PATRL_BORO_NM:
root
 |-- CREATE_DATE: string (nullable = true)
 |-- INCIDENT_DATE: string (nullable = true)
 |-- INCIDENT_TIME: timestamp (nullable = true)
 |-- NYPD_PCT_CD: integer (nullable = true)
 |-- BORO_NM: string (nullable = true)
 |-- GEO_CD_X: integer (nullable = true)
 |-- GEO_CD_Y: integer (nullable = true)
 |-- RADIO_CODE: string (nullable = true)
 |-- TYP_DESC: string (nullable = true)
 |-- CIP_JOBS: string (nullable = true)
 |-- ADD_TS: string (nullable = true)
 |-- DISP_TS: string (nullable = true)
 |-- ARRIVD_TS: string (nullable = true)
 |-- CLOSNG_TS: string (nullable = true)
 |-- Latitude: double (nullable = true)
 |-- Longitude: double (nullable = true)
```

[12]:
```
from pyspark.sql.functions import to_timestamp

# Convert columns to timestamp format
df_cleaned = df_cleaned.withColumn("ADD_TS", to_timestamp("ADD_TS", "MM/dd/yyyy␣
↪hh:mm:ss a")) \
                       .withColumn("DISP_TS", to_timestamp("DISP_TS", "MM/dd/
↪yyyy hh:mm:ss a")) \
                       .withColumn("ARRIVD_TS", to_timestamp("ARRIVD_TS", "MM/
↪dd/yyyy hh:mm:ss a")) \
```

```
                        .withColumn("CLOSNG_TS", to_timestamp("CLOSNG_TS", "MM/
↪dd/yyyy hh:mm:ss a"))

# Verify the schema to ensure correct types
print("Schema after converting to timestamp:")
df_cleaned.printSchema()

# Show sample rows
df_cleaned.select("ADD_TS", "DISP_TS", "ARRIVD_TS", "CLOSNG_TS").show(5,␣
↪truncate=False)
```

```
Schema after converting to timestamp:
root
 |-- CREATE_DATE: string (nullable = true)
 |-- INCIDENT_DATE: string (nullable = true)
 |-- INCIDENT_TIME: timestamp (nullable = true)
 |-- NYPD_PCT_CD: integer (nullable = true)
 |-- BORO_NM: string (nullable = true)
 |-- GEO_CD_X: integer (nullable = true)
 |-- GEO_CD_Y: integer (nullable = true)
 |-- RADIO_CODE: string (nullable = true)
 |-- TYP_DESC: string (nullable = true)
 |-- CIP_JOBS: string (nullable = true)
 |-- ADD_TS: timestamp (nullable = true)
 |-- DISP_TS: timestamp (nullable = true)
 |-- ARRIVD_TS: timestamp (nullable = true)
 |-- CLOSNG_TS: timestamp (nullable = true)
 |-- Latitude: double (nullable = true)
 |-- Longitude: double (nullable = true)

+-------------------+-------------------+-------------------+-------------------
+
|ADD_TS             |DISP_TS            |ARRIVD_TS          |CLOSNG_TS
|
+-------------------+-------------------+-------------------+-------------------
+
|2024-01-01 00:01:21|2024-01-01 00:02:19|2024-01-01 01:19:58|2024-01-01
01:20:02|
|2024-01-01 00:06:11|2024-01-01 00:07:19|2024-01-01 00:19:27|2024-01-01
01:03:22|
|2024-01-01 00:04:51|2024-01-01 00:09:21|2024-01-01 00:15:11|2024-01-01
00:56:56|
|2024-01-01 00:04:57|2024-01-01 00:12:08|2024-01-01 00:29:16|2024-01-01
00:29:53|
|2024-01-01 00:00:07|2024-01-01 00:00:07|2024-01-01 00:00:07|2024-01-01
00:30:23|
+-------------------+-------------------+-------------------+-------------------
+
```

```
only showing top 5 rows
```

[13]:
```python
from pyspark.sql.functions import unix_timestamp, round

# Calculate correct response time differences in minutes
df_cleaned = df_cleaned.withColumn("dispatch_time",␣
 ↪round((unix_timestamp("DISP_TS") - unix_timestamp("ADD_TS")) / 60, 2)) \
                         .withColumn("arrival_time",␣
 ↪round((unix_timestamp("ARRIVD_TS") - unix_timestamp("DISP_TS")) / 60, 2)) \
                         .withColumn("total_response_time",␣
 ↪round((unix_timestamp("ARRIVD_TS") - unix_timestamp("ADD_TS")) / 60, 2))

# Show the corrected metrics
df_cleaned.select("ADD_TS", "DISP_TS", "ARRIVD_TS", "dispatch_time",␣
 ↪"arrival_time", "total_response_time").show(10)
```

```
+-------------------+-------------------+-------------------+-------------+-----
-------+-------------------+
|             ADD_TS|            DISP_TS|
ARRIVD_TS|dispatch_time|arrival_time|total_response_time|
+-------------------+-------------------+-------------------+-------------+-----
-------+-------------------+
|2024-01-01 00:01:21|2024-01-01 00:02:19|2024-01-01 01:19:58|         0.97|
77.65|              78.62|
|2024-01-01 00:06:11|2024-01-01 00:07:19|2024-01-01 00:19:27|         1.13|
12.13|              13.27|
|2024-01-01 00:04:51|2024-01-01 00:09:21|2024-01-01 00:15:11|          4.5|
5.83|              10.33|
|2024-01-01 00:04:57|2024-01-01 00:12:08|2024-01-01 00:29:16|         7.18|
17.13|              24.32|
|2024-01-01 00:00:07|2024-01-01 00:00:07|2024-01-01 00:00:07|          0.0|
0.0|                0.0|
|2024-01-01 00:00:14|2024-01-01 00:08:24|2024-01-01 00:36:32|         8.17|
28.13|               36.3|
|2024-01-01 00:00:25|2024-01-01 00:00:25|2024-01-01 00:00:25|          0.0|
0.0|                0.0|
|2024-01-01 00:00:35|2024-01-01 00:00:35|2024-01-01 00:00:35|          0.0|
0.0|                0.0|
|2024-01-01 00:05:03|2024-01-01 00:15:06|2024-01-01 00:42:21|        10.05|
27.25|               37.3|
|2024-01-01 00:00:51|2024-01-01 00:22:17|2024-01-01 00:30:42|        21.43|
8.42|              29.85|
+-------------------+-------------------+-------------------+-------------+-----
-------+-------------------+
only showing top 10 rows
```

[14]:
```
from pyspark.sql.functions import col

# Identify outliers for dispatch_time and arrival_time using 95th percentile
dispatch_quantile = df_cleaned.approxQuantile("dispatch_time", [0.95], 0.0)[0]
arrival_quantile = df_cleaned.approxQuantile("arrival_time", [0.95], 0.0)[0]

print(f"95th percentile of dispatch_time: {dispatch_quantile}")
print(f"95th percentile of arrival_time: {arrival_quantile}")

# Filter out extreme values for dispatch_time and arrival_time
df_cleaned = df_cleaned.filter(
    (col("dispatch_time") > 0) & (col("dispatch_time") <= dispatch_quantile) &
    (col("arrival_time") > 0) & (col("arrival_time") <= arrival_quantile)
)

# Verify the filtered dataset
df_cleaned.describe("dispatch_time", "arrival_time").show()
```

```
95th percentile of dispatch_time: 20.22
95th percentile of arrival_time: 101.27

[Stage 40:====================================================>    (11 + 1) / 12]

+-------+-----------------+------------------+
|summary|    dispatch_time|      arrival_time|
+-------+-----------------+------------------+
|  count|          1519219|           1519219|
|   mean|3.215696466408669| 16.94440944327612|
| stddev|4.220197489145211|22.136328578888538|
|    min|             0.02|              0.02|
|    max|            20.22|            101.27|
+-------+-----------------+------------------+
```

[15]:
```
# Filter rows with negative response times or any invalid values
df_cleaned = df_cleaned.filter(
    (col("dispatch_time") > 0) &
    (col("arrival_time") > 0) &
    (col("total_response_time") > 0)
)

# Verify cleaned data
print("After filtering invalid values:")
df_cleaned.select("dispatch_time", "arrival_time", "total_response_time").
 ↪describe().show()
```

```
After filtering invalid values:

[Stage 43:==========================================>              (9 + 3) / 12]

+-------+-----------------+------------------+--------------------+
|summary|    dispatch_time|      arrival_time|total_response_time|
+-------+-----------------+------------------+--------------------+
|  count|          1519219|           1519219|             1519219|
|   mean|3.215696466408669| 16.94440944327612| 20.160261772661084|
| stddev|4.220197489145211|22.136328578888538|   23.61289080671234|
|    min|             0.02|              0.02|                0.03|
|    max|            20.22|            101.27|              121.35|
+-------+-----------------+------------------+--------------------+
```

[16]:

```python
from pyspark.ml.feature import StandardScaler, VectorAssembler

# Assemble features for scaling
assembler = VectorAssembler(inputCols=["dispatch_time", "arrival_time"],␣
 ↪outputCol="raw_features")
df_scaled = assembler.transform(df_cleaned)

# Apply StandardScaler
scaler = StandardScaler(inputCol="raw_features", outputCol="features",␣
 ↪withStd=True, withMean=True)
scaler_model = scaler.fit(df_scaled)
df_final = scaler_model.transform(df_scaled).select("features",␣
 ↪"total_response_time")

# Show scaled data
print("Scaled Features:")
df_final.show(5, truncate=False)
```

```
Scaled Features:
+-------------------------------------------+-------------------+
|features                                   |total_response_time|
+-------------------------------------------+-------------------+
|[-0.5321306579097865,2.7423513497455883]   |78.62              |
|[-0.4942177402296219,-0.21748906672196322]|13.27              |
|[0.3043230884088464,-0.5020891067669201]   |10.33              |
|[0.9393644595516045,0.008383980932764353]  |24.32              |
|[1.1739506376976234,0.5053046857731651]    |36.3               |
+-------------------------------------------+-------------------+
only showing top 5 rows
```

[17]:
```python
# Combine all partitions into one
df_cleaned = df_cleaned.coalesce(1)

# Specify the output folder path
output_folder = "gs://nypdbucket/notebooks/jupyter/cleaned_dataset"

# Write the cleaned dataset as a single CSV file
df_cleaned.write.mode("overwrite").option("header", "true").csv(output_folder)

print("Final cleaned dataset saved successfully as a single CSV file!")
```

```
Final cleaned dataset saved successfully as a single CSV file!
```

[18]:
```python
!gsutil mv gs://nypdbucket/notebooks/jupyter/cleaned_dataset/part-00000*.csv gs:
↪//nypdbucket/notebooks/jupyter/cleaned_dataset/final_cleaned_dataset.csv
```

```
Copying gs://nypdbucket/notebooks/jupyter/cleaned_dataset/part-00000-c5d2faff-
641c-4b05-b267-072eb7c90d24-c000.csv [Content-Type=application/octet-stream]…
Removing gs://nypdbucket/notebooks/jupyter/cleaned_dataset/part-00000-c5d2faff-
641c-4b05-b267-072eb7c90d24-c000.csv…

Operation completed over 1 objects/372.5 MiB.
```

[19]:
```python
#https://storage.cloud.google.com/nypdbucket/notebooks/jupyter/nypd_dataset.csv
csv_file_path = "gs://nypdbucket/notebooks/jupyter/cleaned_dataset/
↪final_cleaned_dataset.csv"


# Read the CSV file into a DataFrame
df = (spark.read.format("csv")
      .option("header", "true")  # If the CSV has a header row
      .option("inferSchema", "true")  # Automatically infer the schema
      .load(csv_file_path))


# Show the first few rows of the DataFrame
df.show(5)
```

```
+-----------+-------------+-------------------+-----------+---------+--------+--
------+----------+---------------------+---------+--------------------+------------
-------+--------------------+--------------------+----------+----------+-----------
--+------------+--------------------+
|CREATE_DATE|INCIDENT_DATE|      INCIDENT_TIME|NYPD_PCT_CD|
BORO_NM|GEO_CD_X|GEO_CD_Y|RADIO_CODE|             TYP_DESC|CIP_JOBS|
ADD_TS|              DISP_TS|           ARRIVD_TS|           CLOSNG_TS| Latitude|
Longitude|dispatch_time|arrival_time|total_response_time|
```

```
+-----------+-------------+-------------------+-----------+---------+--------+--
------+----------+--------------------+--------+------------------+------------
-------+-------------------+-------------------+---------+----------+-----------
--+------------+-------------------+
| 01/01/2024|   12/31/2023|2024-12-17 22:31:06|        108|   QUEENS| 1007298|
209993|        53S|VEHICLE ACCIDENT:…| Non CIP|2024-01-01 00:01:21|2024-01-01
00:02:19|2024-01-01 01:19:58|2024-01-01 01:20:02|40.743037|-73.916826|
0.97|       77.65|              78.62|
| 01/01/2024|   12/31/2023|2024-12-17 23:53:22|        114|   QUEENS| 1002279|
222019|       34K1|ASSAULT (IN PROGR…|Critical|2024-01-01 00:06:11|2024-01-01
00:07:19|2024-01-01 00:19:27|2024-01-01 01:03:22|40.776057|-73.934906|
1.13|       12.13|              13.27|
| 01/01/2024|   12/31/2023|2024-12-17 23:59:17|         49|    BRONX| 1020929|
254201|       11C4|ALARMS: COMMERCIA…| Non CIP|2024-01-01 00:04:51|2024-01-01
00:09:21|2024-01-01 00:15:11|2024-01-01 00:56:56| 40.86433|-73.867393|
4.5|        5.83|              10.33|
| 01/01/2024|   12/31/2023|2024-12-17 23:59:30|         34|MANHATTAN| 1003734|
253432|       11C4|ALARMS: COMMERCIA…| Non CIP|2024-01-01 00:04:57|2024-01-01
00:12:08|2024-01-01 00:29:16|2024-01-01 00:29:53|40.862274|-73.929562|
7.18|       17.13|              24.32|
| 01/01/2024|   01/01/2024|2024-12-17 00:00:14|         19|MANHATTAN|  992074|
217827|        53D|VEHICLE ACCIDENT:…| Non CIP|2024-01-01 00:00:14|2024-01-01
00:08:24|2024-01-01 00:36:32|2024-01-01 00:48:57|40.764566|-73.971757|
8.17|       28.13|               36.3|
+-----------+-------------+-------------------+-----------+---------+--------+--
------+----------+--------------------+--------+------------------+------------
-------+-------------------+-------------------+---------+----------+-----------
--+------------+-------------------+
only showing top 5 rows
```

[ ]:

[20]:
```python
counts = df_cleaned.select(
    [(F.sum(F.col(c).isNull().cast("int")).alias(c)) for c in df_cleaned.
 ↪columns]
)

# Show the count of null values for each column
counts.show()
```

```
[Stage 54:>                                                         (0 + 1) / 1]

+-----------+-------------+-------------+-----------+-------+--------+--------+-
---------+--------+--------+------+-------+---------+---------+--------+--------
-+-------------+------------+-------------------+
|CREATE_DATE|INCIDENT_DATE|INCIDENT_TIME|NYPD_PCT_CD|BORO_NM|GEO_CD_X|GEO_CD_Y|R
ADIO_CODE|TYP_DESC|CIP_JOBS|ADD_TS|DISP_TS|ARRIVD_TS|CLOSNG_TS|Latitude|Longitud
e|dispatch_time|arrival_time|total_response_time|
```

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

1.1 Temporal Analysis

[21]:
```
# Modify the INCIDENT_TIME column to extract only the time
df_cleaned_eda = df_cleaned.withColumn("INCIDENT_TIME", F.
 ↪date_format("INCIDENT_TIME", "HH:mm:ss"))\
                            .withColumn("INCIDENT_DATE", F.
 ↪to_date("INCIDENT_DATE", "MM/dd/yyyy"))


# Show the updated DataFrame
df_cleaned.show(1)
```

```
+-----------+-------------+-------------------+-----------+-------+--------+----
----+----------+--------------------+--------+-------------------+--------------
-----+-------------------+-------------------+---------+----------+-------------
+------------+------------------+
|CREATE_DATE|INCIDENT_DATE|
INCIDENT_TIME|NYPD_PCT_CD|BORO_NM|GEO_CD_X|GEO_CD_Y|RADIO_CODE|
TYP_DESC|CIP_JOBS|             ADD_TS|            DISP_TS|          ARRIVD_TS|
CLOSNG_TS| Latitude| Longitude|dispatch_time|arrival_time|total_response_time|
+-----------+-------------+-------------------+-----------+-------+--------+----
----+----------+--------------------+--------+-------------------+--------------
-----+-------------------+-------------------+---------+----------+-------------
+------------+------------------+
| 01/01/2024|   12/31/2023|2024-12-17 22:31:06|        108| QUEENS| 1007298|
209993|       53S|VEHICLE ACCIDENT:…| Non CIP|2024-01-01 00:01:21|2024-01-01
00:02:19|2024-01-01 01:19:58|2024-01-01 01:20:02|40.743037|-73.916826|
0.97|       77.65|              78.62|
+-----------+-------------+-------------------+-----------+-------+--------+----
----+----------+--------------------+--------+-------------------+--------------
-----+-------------------+-------------------+---------+----------+-------------
+------------+------------------+
only showing top 1 row
```

[22]:
```python
# Extract time-related features
df_cleaned_eda = df_cleaned_eda.withColumn("hour", F.hour("INCIDENT_TIME")) \
                        .withColumn("day_of_week", F.dayofweek("INCIDENT_DATE"))␣
 ↪\
                        .withColumn("month", F.month("INCIDENT_DATE"))

# Hourly analysis
df_cleaned_eda.groupBy("hour").count().orderBy("hour").show()

# Day of the week analysis
df_cleaned_eda.groupBy("day_of_week").count().orderBy("day_of_week").show()

# Monthly analysis
df_cleaned_eda.groupBy("month").count().orderBy("month").show()
```

```
+----+-----+
|hour|count|
+----+-----+
|   0|66644|
|   1|59447|
|   2|51055|
|   3|43055|
|   4|40225|
|   5|36937|
|   6|39313|
|   7|46051|
|   8|72157|
|   9|73172|
|  10|71463|
|  11|71298|
|  12|71972|
|  13|71337|
|  14|68378|
|  15|62427|
|  16|81262|
|  17|78908|
|  18|78535|
|  19|76338|
+----+-----+
only showing top 20 rows
```

```
+-----------+------+
|day_of_week| count|
+-----------+------+
```

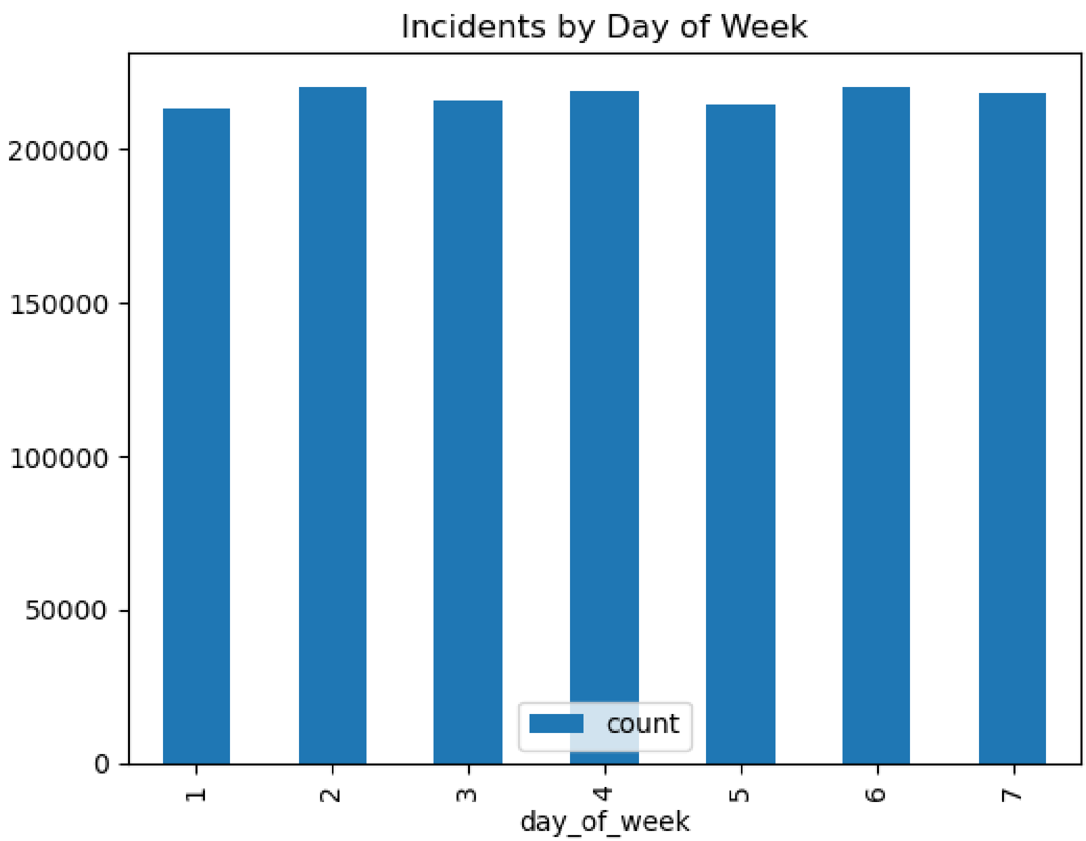
```
|          1|212838|
|          2|220030|
|          3|215403|
|          4|218417|
|          5|214520|
|          6|220142|
|          7|217869|
+-----------+------+
```

```
 [Stage 58:>                                                         (0 + 1) /
1]
```

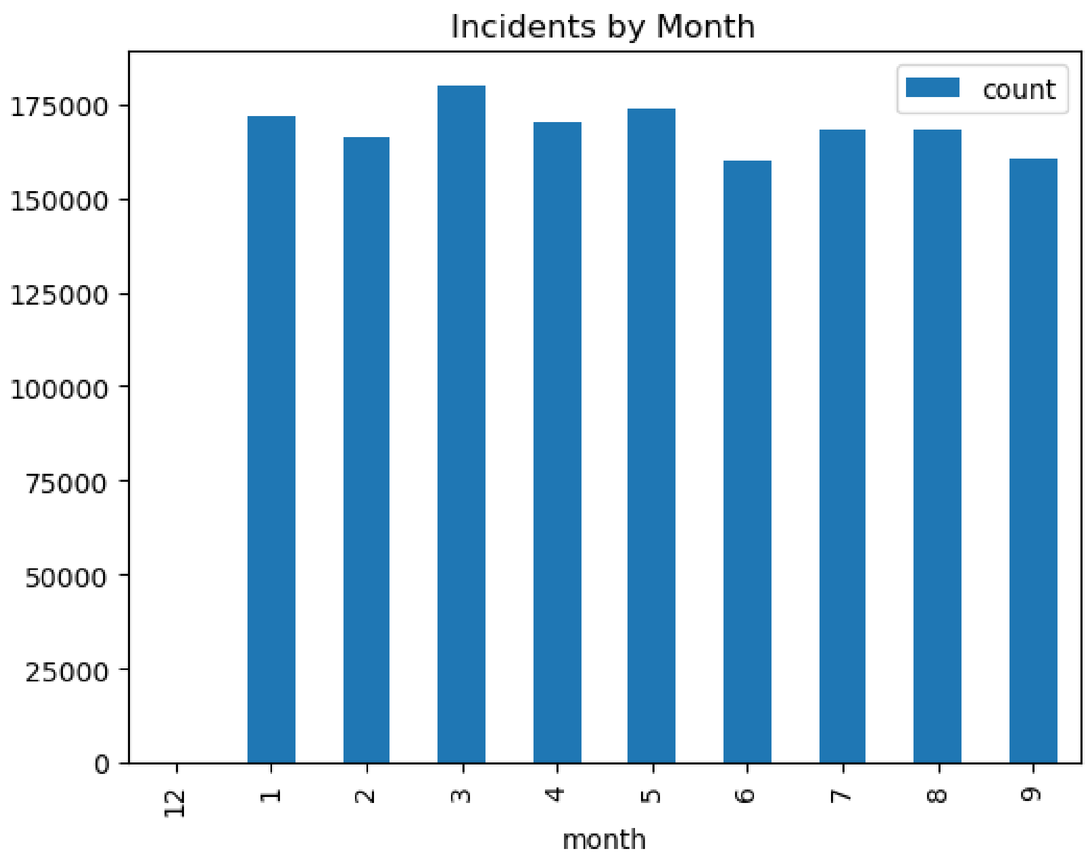
```
+-----+------+
|month| count|
+-----+------+
|    1|171869|
|    2|165935|
|    3|179985|
|    4|170371|
|    5|173827|
|    6|159930|
|    7|168275|
|    8|168419|
|    9|160604|
|   12|     4|
+-----+------+
```

[23]:

```
# Convert to Pandas for visualization
day_of_week_data = df_cleaned_eda.groupBy("day_of_week").count().toPandas()
month_data = df_cleaned_eda.groupBy("month").count().toPandas()

# Plot day of week
day_of_week_data.plot(x="day_of_week", y="count", kind="bar", title="Incidents␣
 ↪by Day of Week")
plt.show()
```



[24]:
```
# Plot month
month_data.plot(x="month", y="count", kind="bar", title="Incidents by Month")
plt.show()
```



[25]:
```
from pyspark.sql.functions import round

# Increase rounding precision for latitude and longitude
df_cleaned_eda = df_cleaned_eda.withColumn("latitude_rounded",␣
↪round("Latitude", 3)) \
                               .withColumn("longitude_rounded", round("Longitude", 3))

# location_data = df_cleaned_eda.groupBy("latitude_rounded",␣
↪"longitude_rounded").count()
# location_data.orderBy("count", ascending=False).show(20, truncate=False)

# Select top 1000 locations with the highest incident counts
top_locations = df_cleaned_eda.groupBy("latitude_rounded", "longitude_rounded").
↪count().orderBy("count", ascending=False).limit(40000).toPandas()
```

[26]:
```
import folium
from folium.plugins import HeatMap
```

```python
# Convert the top_locations DataFrame into a format suitable for HeatMap
heatmap_data = [[row["latitude_rounded"], row["longitude_rounded"],␣
↪row["count"]] for index, row in top_locations.iterrows()]

# Create a base map centered at the mean latitude and longitude
m = folium.Map(
    location=[top_locations["latitude_rounded"].mean(),␣
↪top_locations["longitude_rounded"].mean()],
    zoom_start=12
)

# Add the heatmap layer
HeatMap(heatmap_data, radius=10, blur=15, max_zoom=1).add_to(m)

# Save and display the map
m.save("top_locations_heatmap.html")
m
```

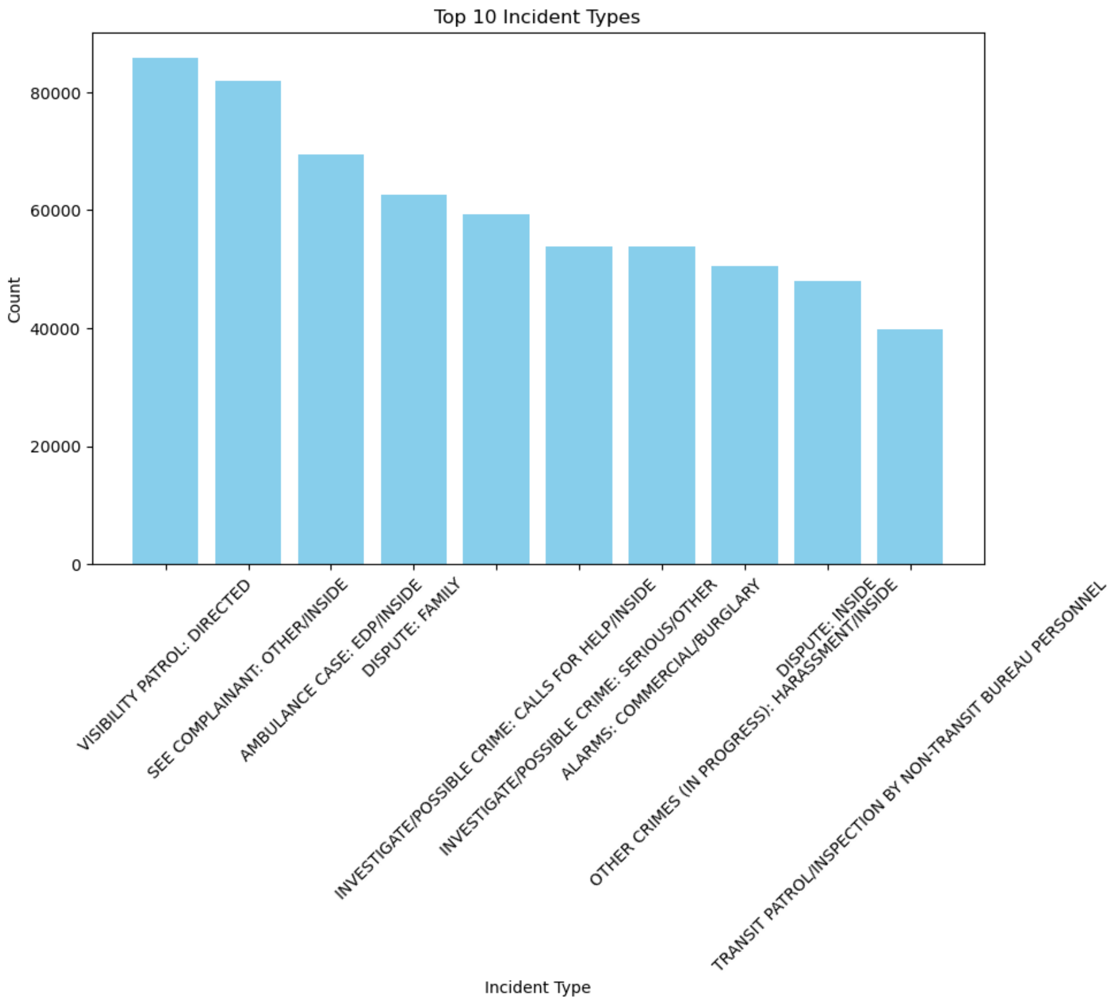
[26]: <folium.folium.Map at 0x7fe0888ac250>

2 EDA FOR INCIDENT CLASSIFICATION

[27]:
```python
# Count incidents by type
incident_type_analysis = df_cleaned_eda.groupBy("TYP_DESC").count().
↪orderBy("count", ascending=False)

# Display the top incident types
incident_type_analysis.show(10, truncate=False)

# Convert incident type analysis to Pandas for plotting
incident_type_pd = incident_type_analysis.limit(10).toPandas()

# Plot the bar chart
plt.figure(figsize=(10, 6))
plt.bar(incident_type_pd["TYP_DESC"], incident_type_pd["count"],␣
↪color="skyblue")
plt.title("Top 10 Incident Types")
plt.xlabel("Incident Type")
plt.ylabel("Count")
plt.xticks(rotation=45)
plt.show()
```

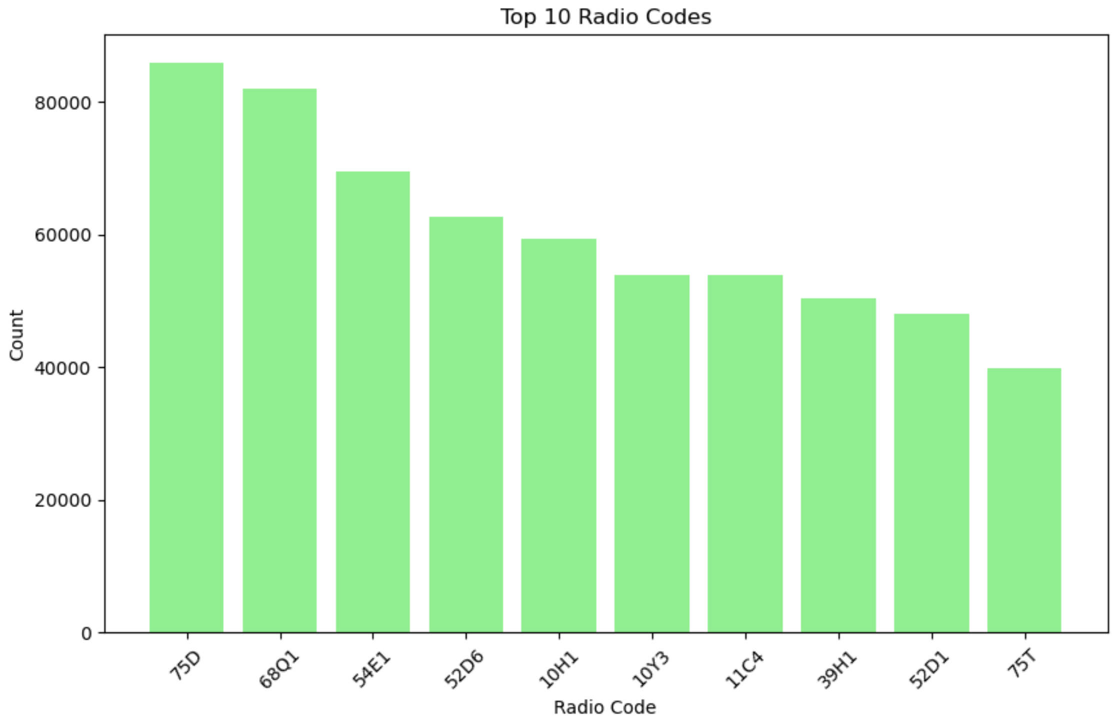
```
+----------------------------------------------------------+-----+
|TYP_DESC                                                  |count|
+----------------------------------------------------------+-----+
```

```
|VISIBILITY PATROL: DIRECTED                                 |85811|
|SEE COMPLAINANT: OTHER/INSIDE                               |81903|
|AMBULANCE CASE: EDP/INSIDE                                  |69473|
|DISPUTE: FAMILY                                             |62591|
|INVESTIGATE/POSSIBLE CRIME: CALLS FOR HELP/INSIDE           |59328|
|INVESTIGATE/POSSIBLE CRIME: SERIOUS/OTHER                   |53826|
|ALARMS: COMMERCIAL/BURGLARY                                 |53799|
|OTHER CRIMES (IN PROGRESS): HARASSMENT/INSIDE               |50462|
|DISPUTE: INSIDE                                             |48050|
|TRANSIT PATROL/INSPECTION BY NON-TRANSIT BUREAU PERSONNEL|39887|
+---------------------------------------------------------+-----+
only showing top 10 rows
```



[28]:
```python
# Count incidents by radio code
radio_code_analysis = df_cleaned_eda.groupBy("RADIO_CODE").count().
↪orderBy("count", ascending=False)

# Display the top radio codes
radio_code_analysis.show(10, truncate=False)

# Convert radio code analysis to Pandas for plotting
radio_code_pd = radio_code_analysis.limit(10).toPandas()

# Plot the bar chart
plt.figure(figsize=(10, 6))
plt.bar(radio_code_pd["RADIO_CODE"], radio_code_pd["count"], color="lightgreen")
plt.title("Top 10 Radio Codes")
plt.xlabel("Radio Code")
plt.ylabel("Count")
plt.xticks(rotation=45)
plt.show()
```

```
+----------+-----+
|RADIO_CODE|count|
+----------+-----+
|75D       |85811|
|68Q1      |81903|
|54E1      |69473|
|52D6      |62591|
|10H1      |59328|
|10Y3      |53826|
|11C4      |53799|
|39H1      |50462|
|52D1      |48050|
|75T       |39887|
+----------+-----+
only showing top 10 rows
```

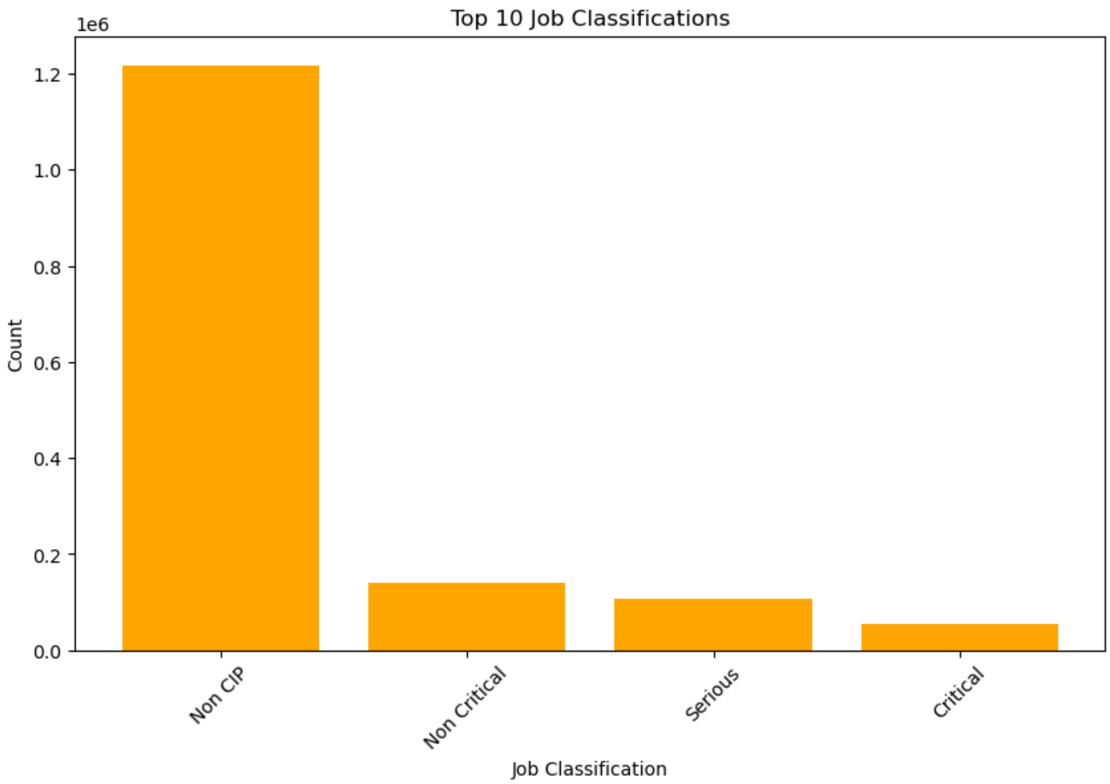


[29]:

```
# Count incidents by job classification
job_classification_analysis = df_cleaned_eda.groupBy("CIP_JOBS").count().
    ↪orderBy("count", ascending=False)

# Display the top job classifications
job_classification_analysis.show(10, truncate=False)

# Convert job classification analysis to Pandas for plotting
job_classification_pd = job_classification_analysis.limit(10).toPandas()

# Plot the bar chart
plt.figure(figsize=(10, 6))
plt.bar(job_classification_pd["CIP_JOBS"], job_classification_pd["count"],␣
    ↪color="orange")
plt.title("Top 10 Job Classifications")
plt.xlabel("Job Classification")
plt.ylabel("Count")
plt.xticks(rotation=45)
plt.show()
```

```
+------------+-------+
|CIP_JOBS    |count  |
```

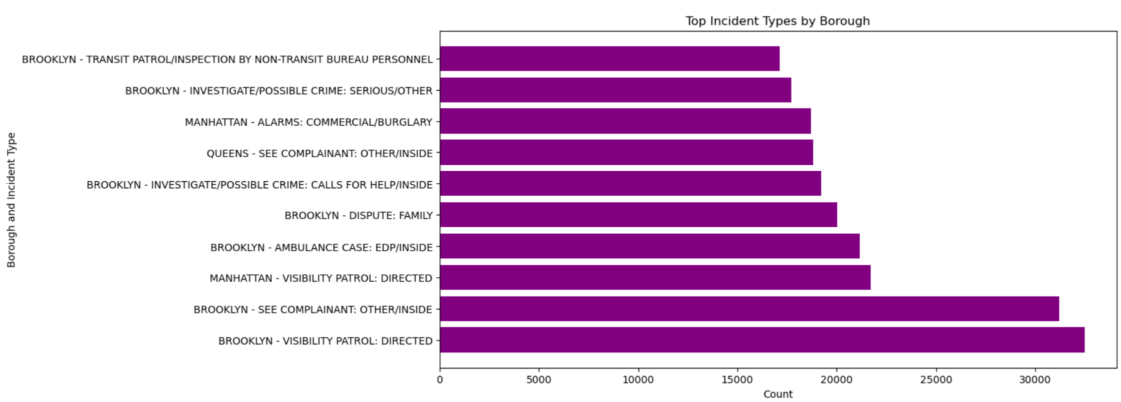
```
+------------+-------+
|Non CIP     |1217682|
|Non Critical|140618 |
|Serious     |106119 |
|Critical    |54800  |
+------------+-------+
```



[30]:
```python
# Group by borough and incident type
incident_types_by_borough = df_cleaned_eda.groupBy("BORO_NM", "TYP_DESC").
 ↪count().orderBy("count", ascending=False)

# Display the top borough-incident type combinations
incident_types_by_borough.show(10, truncate=False)

# Convert to Pandas for plotting
incident_types_pd = incident_types_by_borough.limit(10).toPandas()

# Plot bar chart
plt.figure(figsize=(12, 6))
```

```python
plt.barh(incident_types_pd["BORO_NM"] + " - " + incident_types_pd["TYP_DESC"],
 ↪incident_types_pd["count"], color="purple")
plt.title("Top Incident Types by Borough")
plt.xlabel("Count")
plt.ylabel("Borough and Incident Type")
plt.show()
```

```
+---------+----------------------------------------------------------+-----+
|BORO_NM  |TYP_DESC                                                  |count|
+---------+----------------------------------------------------------+-----+
|BROOKLYN |VISIBILITY PATROL: DIRECTED                               |32476|
|BROOKLYN |SEE COMPLAINANT: OTHER/INSIDE                             |31219|
|MANHATTAN|VISIBILITY PATROL: DIRECTED                               |21704|
|BROOKLYN |AMBULANCE CASE: EDP/INSIDE                                |21156|
|BROOKLYN |DISPUTE: FAMILY                                           |20031|
|BROOKLYN |INVESTIGATE/POSSIBLE CRIME: CALLS FOR HELP/INSIDE         |19206|
|QUEENS   |SEE COMPLAINANT: OTHER/INSIDE                             |18806|
|MANHATTAN|ALARMS: COMMERCIAL/BURGLARY                               |18684|
|BROOKLYN |INVESTIGATE/POSSIBLE CRIME: SERIOUS/OTHER                 |17696|
|BROOKLYN |TRANSIT PATROL/INSPECTION BY NON-TRANSIT BUREAU PERSONNEL|17104|
+---------+----------------------------------------------------------+-----+
only showing top 10 rows
```



[31]:
```python
# Average response times by borough
response_time_by_borough = df_cleaned_eda.groupBy("BORO_NM").
 ↪mean("dispatch_time", "arrival_time", "total_response_time")
response_time_by_borough.show(truncate=False)
```

```
 [Stage 70:>                                                        (0 + 1) /
1]
```

```
+-------------+------------------+------------------+----------------------+
|BORO_NM      |avg(dispatch_time)|avg(arrival_time) |avg(total_response_time)|
+-------------+------------------+------------------+----------------------+
|QUEENS       |3.4332315183604516|19.172429105356045|22.60586888162176     |
|BRONX        |3.742271012683865 |17.651551254424962|21.393927382916427    |
|MANHATTAN    |3.0732346362753376|15.06390266740991 |18.1371637638107      |
|BROOKLYN     |2.9635499758643937|16.63173771245205 |19.595526404670345    |
|STATEN ISLAND|2.848513425194413 |15.632137955451899|18.480836147269635    |
|(null)       |1.568095238095238 |13.253333333333334|14.820952380952384    |
+-------------+------------------+------------------+----------------------+
```

[32]:

```python
from pyspark.sql.functions import hour, dayofweek

# Add hour and day of the week columns
df_cleaned_eda = df_cleaned_eda.withColumn("hour", hour("INCIDENT_TIME")) \
                               .withColumn("day_of_week", dayofweek("INCIDENT_DATE"))

# Average response times by hour
response_time_by_hour = df_cleaned_eda.groupBy("hour").mean("dispatch_time", "arrival_time", "total_response_time")
response_time_by_hour.show(truncate=False)

# Average response times by day of the week
response_time_by_day = df_cleaned_eda.groupBy("day_of_week").mean("dispatch_time", "arrival_time", "total_response_time")
response_time_by_day.show(truncate=False)
```

```
+----+------------------+------------------+----------------------+
|hour|avg(dispatch_time)|avg(arrival_time) |avg(total_response_time)|
+----+------------------+------------------+----------------------+
|22  |3.1272606028272865|13.53621373542937 |16.66363704831049     |
|23  |4.341739894208651 |18.10833291043063 |22.450261354985987    |
|0   |2.7815602304781795|14.271719284559152|17.05347188043951     |
|1   |2.467993675038265 |12.777316433127371|15.245506922132085    |
|2   |2.3968036431300566|12.546193712664378|14.94316325531275     |
|3   |2.4486705376843902|12.623134130764058|15.07200092904415     |
|4   |2.4914846488502476|13.001222871349647|15.492924549409468    |
|5   |2.507468121395905 |13.641176868723159|16.148846684895787    |
|6   |2.6962617963524047|14.082330018061256|16.77873069976859     |
|7   |4.150677509717355 |23.767868667348697|27.918648237823497    |
|8   |2.567756004268294 |18.62475393932958 |21.192663220479318    |
|9   |2.860873968184314 |18.695965533266726|21.55695238616037     |
|10  |3.125105299245533 |18.78806711165481 |21.913313322979636    |
```

```
|11  |3.2958690285840233|19.24268983702451 |22.538693511740846      |
|12  |3.4051720113375272|19.536395959542755|22.94173150669866       |
|13  |3.3526879459464434|18.077859876364244|21.43069893603734       |
|14  |3.3539938284241235|15.20779753722193 |18.561966129457435      |
|15  |4.8294797123040105|20.968853380750698|25.798443622151325      |
|16  |3.327067386970393 |17.853575964168037|21.180765302356836      |
|17  |3.28389922441303  |17.46224476605933 |20.746284533888502      |
+----+------------------+------------------+------------------------+
only showing top 20 rows

 [Stage 72:>                                                         (0 + 1) /
1]

+-----------+------------------+------------------+------------------------+
|day_of_week|avg(dispatch_time)|avg(arrival_time) |avg(total_response_time)|
+-----------+------------------+------------------+------------------------+
|1          |3.2058363638085847|16.66860969375259 |19.87459903776274       |
|2          |3.2196181884301733|17.178221288001104|20.39801754306018       |
|3          |3.1854086990445323|16.848214973788462|20.03376726414887       |
|4          |3.1067741064125314|16.789266403250615|19.89619612941861       |
|5          |3.23745730002018  |16.945285148232344|20.18288635091962       |
|6          |3.3220365491380233|17.247511106462397|20.569702146793453      |
|7          |3.2316336881350463|16.921222294125407|20.15301777673505       |
+-----------+------------------+------------------+------------------------+
```

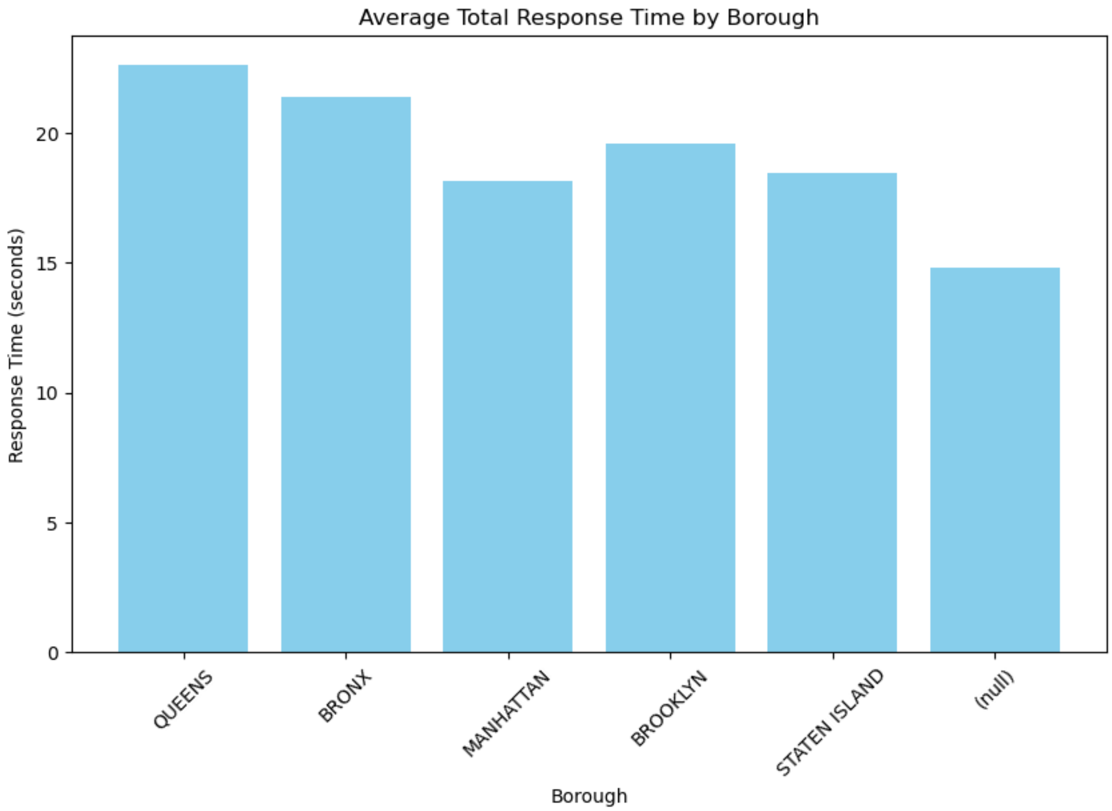
[33]:
```python
# Average response times by incident type
response_time_by_incident = df_cleaned_eda.groupBy("TYP_DESC").
 ↪mean("dispatch_time", "arrival_time", "total_response_time")
response_time_by_incident.orderBy("avg(total_response_time)", ascending=False).
 ↪show(10, truncate=False)
```

```
 [Stage 73:>                                                         (0 + 1) /
1]

+-----------------------------------------------------+------------------+--------
----------+------------------------+
|TYP_DESC
|avg(dispatch_time)|avg(arrival_time) |avg(total_response_time)|
+-----------------------------------------------------+------------------+--------
----------+------------------------+
|CELLULAR OPEN LINE                                   |14.65             |60.17
|74.82                   |
|LARCENY (PAST): VEHICLE/SCHOOL                       |7.983333333333333
|50.50666666666666 |58.48666666666666       |
|DISORDERLY: NOISE/OUTSIDE                            |8.43              |46.18
|54.62                   |
```

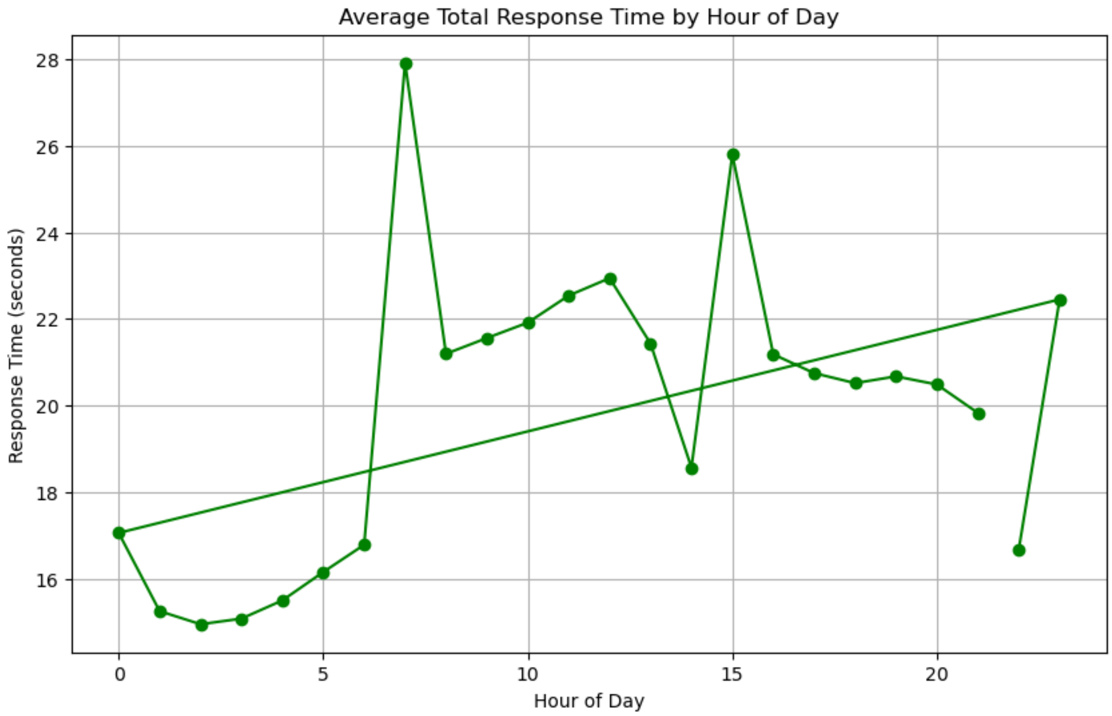
```
|UTILITY TROUBLE (SPECIFY): LTD ACC HWY            |1.2               |52.85
|54.05                    |
|ASSAULT (PAST): OTHER/LTD ACC HWY                 |8.233333333333333
|41.54333333333333 |49.776666666666664       |
|LARCENY (PAST): OTHER/LTD ACC HWY                 |8.540000000000001 |39.725
|48.26500000000001        |
|SUSP PACKAGE: LTD ACC HWY                         |6.823333333333334
|41.22166666666667 |48.04500000000001        |
|INVESTIGATE/POSSIBLE CRIME: NARCO SALES/LTD ACC HWY|2.58              |45.2
|47.78                    |
|DISABLED VEHICLE: LTD ACC HWY                     |8.128235294117646
|38.980000000000004|47.107058823529414       |
|INVESTIGATE/POSSIBLE CRIME: SUSP VEHICLE/TRANSIT   |4.0525
|42.620000000000005|46.675                   |
+--------------------------------------------------+------------------+--------
----------+-----------------------+
only showing top 10 rows
```

[34]:

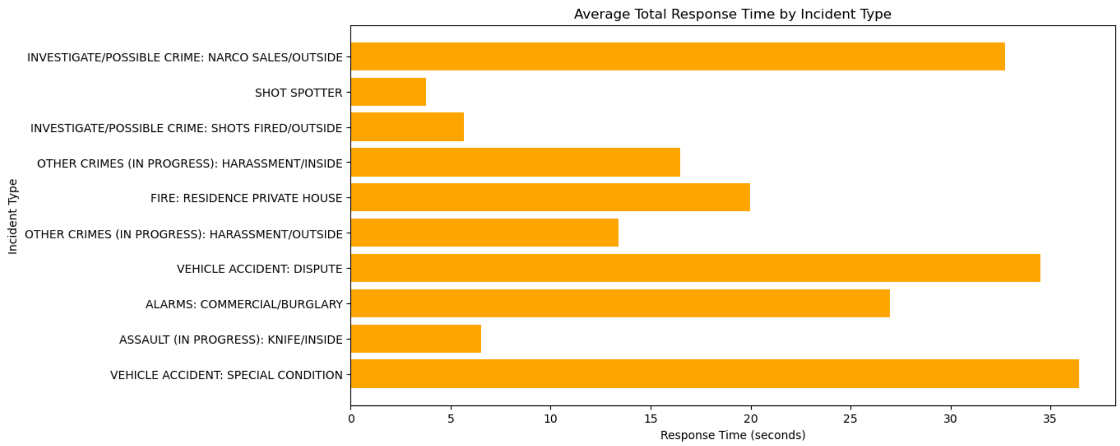
```python
import matplotlib.pyplot as plt

# Convert to Pandas for plotting
response_time_borough_pd = response_time_by_borough.toPandas()

# Plot bar chart
plt.figure(figsize=(10, 6))
plt.bar(response_time_borough_pd["BORO_NM"],
 ↪response_time_borough_pd["avg(total_response_time)"], color="skyblue")
plt.title("Average Total Response Time by Borough")
plt.xlabel("Borough")
plt.ylabel("Response Time (seconds)")
plt.xticks(rotation=45)
plt.show()
```



[35]:
```
# Convert to Pandas for plotting
response_time_hour_pd = response_time_by_hour.toPandas()

# Plot line chart
plt.figure(figsize=(10, 6))
plt.plot(response_time_hour_pd["hour"],␣
↪response_time_hour_pd["avg(total_response_time)"], marker="o", color="green")
plt.title("Average Total Response Time by Hour of Day")
plt.xlabel("Hour of Day")
plt.ylabel("Response Time (seconds)")
plt.grid()
plt.show()
```



[36]:
```
# Convert to Pandas for plotting
response_time_incident_pd = response_time_by_incident.limit(10).toPandas()

# Plot horizontal bar chart
plt.figure(figsize=(12, 6))
plt.barh(response_time_incident_pd["TYP_DESC"],␣
↪response_time_incident_pd["avg(total_response_time)"], color="orange")
plt.title("Average Total Response Time by Incident Type")
plt.xlabel("Response Time (seconds)")
plt.ylabel("Incident Type")
plt.show()
```



3 Modeling

3.1 Linear Regression

[37]:
```python
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.sql import SparkSession
from pyspark.ml.regression import GBTRegressor
```

[38]:
```python
# Feature columns for the regression model
feature_columns = ["dispatch_time", "arrival_time"]

# Assemble features into a single vector
assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")

# Transform the data
df_prepared = assembler.transform(df_cleaned).select("features",␣
↪"total_response_time")

# Show the prepared data
print("Prepared Data for Linear Regression:")
df_prepared.show(5, truncate=False)
```

```
Prepared Data for Linear Regression:
+------------+-------------------+
|features    |total_response_time|
+------------+-------------------+
|[0.97,77.65]|78.62              |
|[1.13,12.13]|13.27              |
|[4.5,5.83]  |10.33              |
|[7.18,17.13]|24.32              |
```

```
|[8.17,28.13]|36.3               |
+------------+-------------------+
only showing top 5 rows
```

[39]:
```python
# Split the data into train and test sets (80% train, 20% test)
train_data, test_data = df_prepared.randomSplit([0.6, 0.4], seed=42)

print("Training Data Count:", train_data.count())
print("Test Data Count:", test_data.count())
```

```
Training Data Count: 912778

[Stage 79:>                                                          (0 + 1) /
1]

Test Data Count: 606441
```

[40]:
```python
# Initialize the Linear Regression model
lr = LinearRegression(featuresCol="features", labelCol="total_response_time",␣
 ↪predictionCol="prediction")

# Train the model
lr_model = lr.fit(train_data)

# Print model coefficients and intercept
print("Coefficients:", lr_model.coefficients)
print("Intercept:", lr_model.intercept)
```

```
24/12/17 01:37:02 WARN Instrumentation: [901bbe0c] regParam is zero, which might
cause numerical instability and overfitting.
[Stage 81:>                                                          (0 + 1) / 1]

Coefficients: [0.9999813888156942,0.9999967289283758]
Intercept: 0.00026806356868136423
```

[41]:
```python
# Predict on test data
predictions = lr_model.transform(test_data)

# Show predictions
print("Linear Regression Predictions:")
predictions.select("features", "total_response_time", "prediction").show(10,␣
 ↪truncate=False)

# Evaluate the model
```

```
evaluator = RegressionEvaluator(labelCol="total_response_time",␣
 ↪predictionCol="prediction", metricName="rmse")

# Calculate RMSE
rmse = evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE):", rmse)

# Optionally calculate R2
evaluator_r2 = RegressionEvaluator(labelCol="total_response_time",␣
 ↪predictionCol="prediction", metricName="r2")
r2 = evaluator_r2.evaluate(predictions)
print("R2 Score:", r2)
```

```
Linear Regression Predictions:


+-----------+-------------------+-------------------+
|features   |total_response_time|prediction         |
+-----------+-------------------+-------------------+
|[0.02,0.02]|0.03               |0.04026762592356276|
|[0.02,0.02]|0.03               |0.04026762592356276|
|[0.02,0.02]|0.03               |0.04026762592356276|
|[0.02,0.02]|0.03               |0.04026762592356276|
|[0.02,0.02]|0.03               |0.04026762592356276|
|[0.02,0.02]|0.03               |0.04026762592356276|
|[0.02,0.02]|0.03               |0.04026762592356276|
|[0.02,0.02]|0.03               |0.04026762592356276|
|[0.02,0.02]|0.03               |0.04026762592356276|
|[0.02,0.02]|0.03               |0.04026762592356276|
+-----------+-------------------+-------------------+
only showing top 10 rows



Root Mean Squared Error (RMSE): 0.004880453984611745

 [Stage 84:>                                                          (0 + 1) /
1]

R2 Score: 0.9999999572697764
```

[42]:
```
# Model summary
training_summary = lr_model.summary

print("Training Summary:")
print("RMSE on Training Data:", training_summary.rootMeanSquaredError)
print("R2 on Training Data:", training_summary.r2)
```

```
Training Summary:
RMSE on Training Data: 0.00488897638958177
R2 on Training Data: 0.9999999571389353
```

3.2 Random Forest

[43]:
```python
from pyspark.ml.regression import RandomForestRegressor

# Initialize Random Forest Regressor
rf = RandomForestRegressor(featuresCol="features",␣
 ↪labelCol="total_response_time", predictionCol="prediction", numTrees=100)

# Train the model
rf_model = rf.fit(train_data)

print("Random Forest model training completed!")
```

```
Random Forest model training completed!
```

[44]:
```python
# Predict on the test data
predictions = rf_model.transform(test_data)

# Show predictions
print("Predictions on Test Data:")
predictions.select("features", "total_response_time", "prediction").show(10,␣
 ↪truncate=False)
```

```
Predictions on Test Data:

WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.spark.util.SizeEstimator$
(file:/usr/lib/spark/jars/spark-core_2.12-3.5.1.jar) to field
java.nio.charset.Charset.name
WARNING: Please consider reporting this to the maintainers of
org.apache.spark.util.SizeEstimator$
WARNING: Use --illegal-access=warn to enable warnings of further illegal
reflective access operations
WARNING: All illegal access operations will be denied in a future release
[Stage 99:>                                                         (0 + 1) / 1]

+-----------+-------------------+------------------+
|features   |total_response_time|prediction        |
+-----------+-------------------+------------------+
|[0.02,0.02]|0.03               |0.7888153334353398|
|[0.02,0.02]|0.03               |0.7888153334353398|
|[0.02,0.02]|0.03               |0.7888153334353398|
|[0.02,0.02]|0.03               |0.7888153334353398|
|[0.02,0.02]|0.03               |0.7888153334353398|
```

```
|[0.02,0.02]|0.03              |0.7888153334353398|
|[0.02,0.02]|0.03              |0.7888153334353398|
|[0.02,0.02]|0.03              |0.7888153334353398|
|[0.02,0.02]|0.03              |0.7888153334353398|
|[0.02,0.02]|0.03              |0.7888153334353398|
+-----------+------------------+------------------+
only showing top 10 rows
```

[45]:

```
# Initialize evaluators
evaluator_rmse = RegressionEvaluator(labelCol="total_response_time",␣
 ↪predictionCol="prediction", metricName="rmse")
evaluator_r2 = RegressionEvaluator(labelCol="total_response_time",␣
 ↪predictionCol="prediction", metricName="r2")

# Calculate RMSE and R²
rmse = evaluator_rmse.evaluate(predictions)
r2 = evaluator_r2.evaluate(predictions)

print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R2 Score: {r2}")
```

```
[Stage 101:>                                                        (0 + 1) / 1]

Root Mean Squared Error (RMSE): 3.678746451618793
R2 Score: 0.9757219146809109
```

3.3 Gradient Boosting

[46]:

```
# Initialize Gradient Boosted Tree Regressor
gbt = GBTRegressor(featuresCol="features", labelCol="total_response_time",␣
 ↪predictionCol="prediction", maxIter=100)

# Train the model
gbt_model = gbt.fit(train_data)

print("Gradient Boosting model training completed!")
```

```
Gradient Boosting model training completed!
```

[47]:

```
# Predict on the test data
predictions = gbt_model.transform(test_data)

# Show predictions
```

```
print("Predictions on Test Data:")
predictions.select("features", "total_response_time", "prediction").show(10,␣
↪truncate=False)
```

```
Predictions on Test Data:

[Stage 1106:>                                                       (0 + 1) /
1]

+-----------+-------------------+--------------------+
|features   |total_response_time|prediction          |
+-----------+-------------------+--------------------+
|[0.02,0.02]|0.03               |0.032046177496824646|
|[0.02,0.02]|0.03               |0.032046177496824646|
|[0.02,0.02]|0.03               |0.032046177496824646|
|[0.02,0.02]|0.03               |0.032046177496824646|
|[0.02,0.02]|0.03               |0.032046177496824646|
|[0.02,0.02]|0.03               |0.032046177496824646|
|[0.02,0.02]|0.03               |0.032046177496824646|
|[0.02,0.02]|0.03               |0.032046177496824646|
|[0.02,0.02]|0.03               |0.032046177496824646|
|[0.02,0.02]|0.03               |0.032046177496824646|
+-----------+-------------------+--------------------+
only showing top 10 rows
```

[48]:
```
# Initialize evaluators
evaluator_rmse = RegressionEvaluator(labelCol="total_response_time",␣
↪predictionCol="prediction", metricName="rmse")
evaluator_r2 = RegressionEvaluator(labelCol="total_response_time",␣
↪predictionCol="prediction", metricName="r2")

# Calculate RMSE and R²
rmse = evaluator_rmse.evaluate(predictions)
r2 = evaluator_r2.evaluate(predictions)

print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R2 Score: {r2}")
```

```
[Stage 1108:>                                                     (0 + 1) / 1]

Root Mean Squared Error (RMSE): 1.5631399430441817
R2 Score: 0.9956166066697358
```

[49]:
```
from pyspark.sql.functions import when, col
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

```python
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.feature import VectorAssembler

# Step 1: Categorize total_response_time into binary classes
threshold = 60  # Set a threshold for classification
df_classification = df_cleaned.withColumn("label",␣
↪when(col("total_response_time") > threshold, 1).otherwise(0))

# Assemble features
feature_columns = ["dispatch_time", "arrival_time"]  # Include relevant features
assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")
df_prepared = assembler.transform(df_classification).select("features", "label")

# Split into train and test sets
train_data, test_data = df_prepared.randomSplit([0.6, 0.4], seed=42)
```

[50]:
```python
# Initialize Random Forest Classifier
rf_classifier = RandomForestClassifier(featuresCol="features",␣
↪labelCol="label", predictionCol="prediction", numTrees=100)

# Train the classifier
rf_model = rf_classifier.fit(train_data)

# Make predictions on test data
predictions = rf_model.transform(test_data)

# Show predictions
predictions.select("features", "label", "prediction").show(10)
```

```
[Stage 1124:>                                                       (0 + 1) / 1]

+-----------+-----+----------+
|   features|label|prediction|
+-----------+-----+----------+
|[0.02,0.02]|    0|       0.0|
|[0.02,0.02]|    0|       0.0|
|[0.02,0.02]|    0|       0.0|
|[0.02,0.02]|    0|       0.0|
|[0.02,0.02]|    0|       0.0|
|[0.02,0.02]|    0|       0.0|
|[0.02,0.02]|    0|       0.0|
|[0.02,0.02]|    0|       0.0|
|[0.02,0.02]|    0|       0.0|
|[0.02,0.02]|    0|       0.0|
+-----------+-----+----------+
only showing top 10 rows
```

[51]:
```python
# Accuracy
evaluator_accuracy = MulticlassClassificationEvaluator(labelCol="label",␣
↪predictionCol="prediction", metricName="accuracy")
accuracy = evaluator_accuracy.evaluate(predictions)

# Precision
evaluator_precision = MulticlassClassificationEvaluator(labelCol="label",␣
↪predictionCol="prediction", metricName="weightedPrecision")
precision = evaluator_precision.evaluate(predictions)

# Recall
evaluator_recall = MulticlassClassificationEvaluator(labelCol="label",␣
↪predictionCol="prediction", metricName="weightedRecall")
recall = evaluator_recall.evaluate(predictions)

# Print the metrics
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
```

```
[Stage 1129:>                                                       (0 + 1) / 1]

Accuracy: 0.9904821738635745
Precision: 0.9903975325539243
Recall: 0.9904821738635745
```

3.4 K-Means clustering

[52]:
```python
from pyspark.ml.clustering import KMeans
from pyspark.ml.feature import VectorAssembler
from pyspark.sql.functions import col
```

[53]:
```python
# Select relevant columns (latitude and longitude)
selected_features = ["Latitude", "Longitude"]  # Add other relevant features if␣
↪needed

# Assemble features into a single vector
assembler = VectorAssembler(inputCols=selected_features, outputCol="features")

# Transform the data
df_prepared = assembler.transform(df_cleaned).select("features",␣
↪*selected_features)

# Show prepared data
print("Prepared Data for K-means Clustering:")
df_prepared.show(5, truncate=False)
```

```
Prepared Data for K-means Clustering:
+----------------------+---------+----------+
|features              |Latitude |Longitude |
+----------------------+---------+----------+
|[40.743037,-73.916826]|40.743037|-73.916826|
|[40.776057,-73.934906]|40.776057|-73.934906|
|[40.86433,-73.867393] |40.86433 |-73.867393|
|[40.862274,-73.929562]|40.862274|-73.929562|
|[40.764566,-73.971757]|40.764566|-73.971757|
+----------------------+---------+----------+
only showing top 5 rows
```

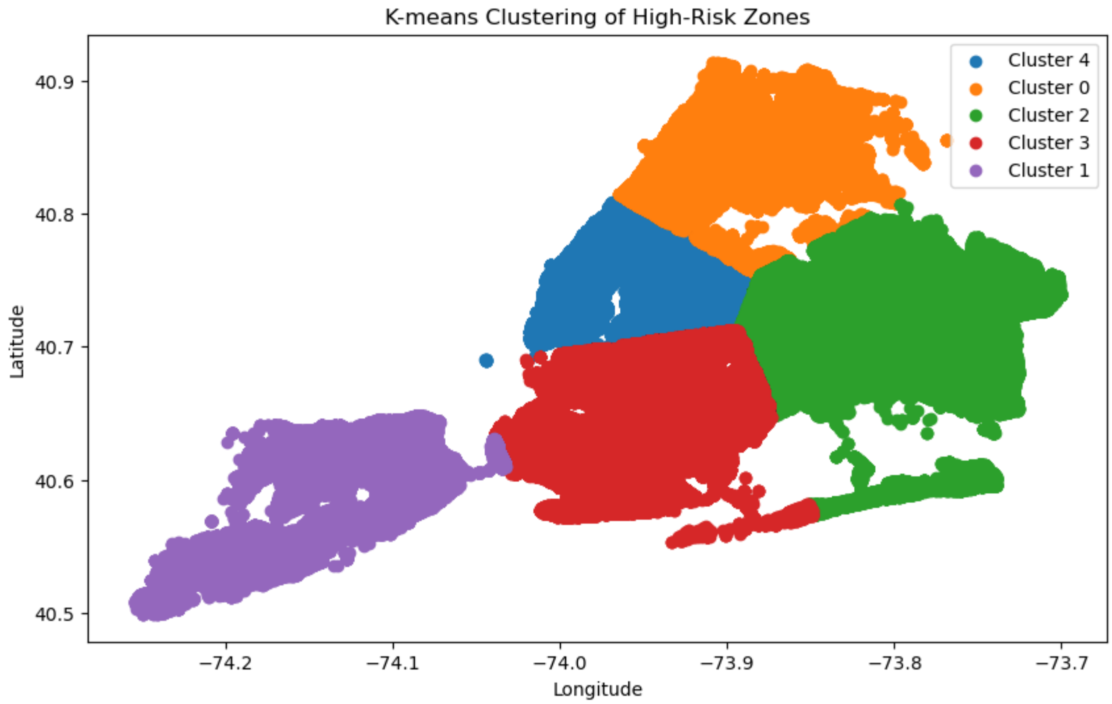
[54]:
```python
# Initialize K-means model
k = 5  # Define the number of clusters
kmeans = KMeans(featuresCol="features", predictionCol="cluster", k=k, seed=42)

# Train the K-means model
kmeans_model = kmeans.fit(df_prepared)

# Assign clusters to data points
df_clusters = kmeans_model.transform(df_prepared)

# Show the resulting clusters
print("Data with Cluster Assignments:")
df_clusters.select("Latitude", "Longitude", "cluster").show(10)
```

```
Data with Cluster Assignments:
+---------+----------+-------+
| Latitude| Longitude|cluster|
+---------+----------+-------+
|40.743037|-73.916826|      4|
|40.776057|-73.934906|      4|
| 40.86433|-73.867393|      0|
|40.862274|-73.929562|      0|
|40.764566|-73.971757|      4|
|40.706102|-73.793242|      2|
|40.740547|-74.008547|      4|
|40.706528|-73.791997|      2|
|40.770813|-73.811147|      2|
| 40.70215|-73.790564|      2|
+---------+----------+-------+
only showing top 10 rows
```

[55]:
```
# Get cluster centers
centers = kmeans_model.clusterCenters()
print("Cluster Centers (High-Risk Zones):")
for idx, center in enumerate(centers):
    print(f"Cluster {idx}: {center}")
```

```
Cluster Centers (High-Risk Zones):
Cluster 0: [ 40.83743223 -73.90282894]
Cluster 1: [ 40.60199467 -74.12495999]
Cluster 2: [ 40.70022946 -73.81260808]
Cluster 3: [ 40.65558713 -73.95103755]
Cluster 4: [ 40.74828972 -73.96746351]
```

[56]:
```
import matplotlib.pyplot as plt

# Convert to Pandas for plotting
df_pandas = df_clusters.select("Latitude", "Longitude", "cluster").toPandas()

# Plot clusters
plt.figure(figsize=(10, 6))
for cluster_id in df_pandas["cluster"].unique():
    cluster_data = df_pandas[df_pandas["cluster"] == cluster_id]
    plt.scatter(cluster_data["Longitude"], cluster_data["Latitude"],␣
 ↪label=f"Cluster {cluster_id}")

plt.title("K-means Clustering of High-Risk Zones")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.legend()
plt.show()
```



[57]:
```
from pyspark.sql.functions import col, sqrt, pow, mean
from pyspark.ml.linalg import Vectors
import numpy as np
```

[58]:
```
from pyspark.sql.functions import col, udf
from pyspark.sql.types import DoubleType
from pyspark.ml.linalg import DenseVector
import math
from pyspark import SparkContext

# Step 2.1: Get cluster centers
centers = kmeans_model.clusterCenters()
cluster_centers = {i: centers[i] for i in range(len(centers))}

# Step 2.2: Broadcast cluster centers for efficiency
sc = SparkContext.getOrCreate()
broadcast_centers = sc.broadcast(cluster_centers)

# Step 2.3: Define a UDF to calculate Euclidean distance
def euclidean_distance(point, cluster_id):
    center = broadcast_centers.value[cluster_id]  # Fetch cluster center from␣
 ↪broadcasted dictionary
    return float(math.sqrt(sum((point[i] - center[i]) ** 2 for i in␣
 ↪range(len(center)))))
```

```
# Register the UDF
distance_udf = udf(euclidean_distance, DoubleType())

# Step 2.4: Add intra-cluster distances to the DataFrame
df_with_distance = df_clusters.withColumn(
    "intra_cluster_distance",
    distance_udf(col("features"), col("cluster"))
)

# Show the updated DataFrame
print("Data with Intra-Cluster Distances:")
df_with_distance.select("features", "cluster", "intra_cluster_distance").
 ↪show(10, truncate=False)
```

```
Data with Intra-Cluster Distances:

 [Stage 1184:>                                                     (0 + 1) /
1]

+----------------------+-------+----------------------+
|features              |cluster|intra_cluster_distance|
+----------------------+-------+----------------------+
|[40.743037,-73.916826]|4      |0.050909216914502266  |
|[40.776057,-73.934906]|4      |0.04279034383589714   |
|[40.86433,-73.867393] |0      |0.04448815501335516   |
|[40.862274,-73.929562]|0      |0.036493420205705035  |
|[40.764566,-73.971757]|4      |0.016833048236449075  |
|[40.706102,-73.793242]|2      |0.02023688849743139   |
|[40.740547,-74.008547]|4      |0.041806732023081376  |
|[40.706528,-73.791997]|2      |0.0215519858106137    |
|[40.770813,-73.811147]|2      |0.07059866101884622   |
|[40.70215,-73.790564] |2      |0.022127579853446676  |
+----------------------+-------+----------------------+
only showing top 10 rows
```

[59]:
```
import numpy as np

# Function to compute pairwise distances between cluster centers
def pairwise_distances(centers):
    k = len(centers)
    distances = np.zeros((k, k))
    for i in range(k):
        for j in range(k):
            if i != j:
```

```
                distances[i][j] = np.linalg.norm(np.array(centers[i]) - np.
 ↪array(centers[j]))
    return distances

# Compute pairwise distances
inter_cluster_distances = pairwise_distances(centers)

# Print pairwise inter-cluster distances
print("Inter-Cluster Distances:")
print(inter_cluster_distances)
```

```
Inter-Cluster Distances:
[[0.         0.32368665 0.16420842 0.18812685 0.11010911]
 [0.32368665 0.         0.32743517 0.18199222 0.21495902]
 [0.16420842 0.32743517 0.         0.14544984 0.16214189]
 [0.18812685 0.18199222 0.14544984 0.         0.09414659]
 [0.11010911 0.21495902 0.16214189 0.09414659 0.        ]]
```

[60]:

```
# Collect intra-cluster dispersions
dispersion_values = df_with_distance.groupBy("cluster").
 ↪agg(mean("intra_cluster_distance").alias("dispersion")).collect()

# Convert dispersion values into a dictionary
dispersion_dict = {row['cluster']: row['dispersion'] for row in␣
 ↪dispersion_values}

# Compute Davies-Bouldin Index
db_index = 0
k = len(centers)

for i in range(k):
    max_ratio = 0
    for j in range(k):
        if i != j:
            ratio = (dispersion_dict[i] + dispersion_dict[j]) /␣
 ↪inter_cluster_distances[i][j]
            max_ratio = max(max_ratio, ratio)
    db_index += max_ratio

db_index /= k

print(f"Davies-Bouldin Index (DBI): {db_index}")
```

```
 [Stage 1185:>                                                       (0 + 1) /
1]

Davies-Bouldin Index (DBI): 0.7477194917382969
```

[61]:
```python
from pyspark.ml.evaluation import ClusteringEvaluator
from pyspark.ml.clustering import KMeans
from pyspark.ml.feature import VectorAssembler

# Step 1: Assemble Features (if not already done)
feature_columns = ["Latitude", "Longitude"]  # Include spatial or relevant␣
↪features
assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")
df_prepared = assembler.transform(df_cleaned)

# Step 2: Train K-means Model
k = 5  # Number of clusters
kmeans = KMeans(featuresCol="features", predictionCol="prediction", k=k,␣
↪seed=42)
kmeans_model = kmeans.fit(df_prepared)

# Step 3: Assign Clusters to Data Points
df_clusters = kmeans_model.transform(df_prepared)

# Step 4: Compute Silhouette Score
evaluator = ClusteringEvaluator(featuresCol="features",␣
↪predictionCol="prediction", metricName="silhouette",␣
↪distanceMeasure="squaredEuclidean")

silhouette_score = evaluator.evaluate(df_clusters)

print(f"Silhouette Score: {silhouette_score}")
```

```
[Stage 1240:>                                                       (0 + 1) / 1]
```

```
Silhouette Score: 0.5913598921815876
```

4 Scale in Scale out

[62]:
```python
import time
from pyspark.ml.regression import RandomForestRegressor, GBTRegressor
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.evaluation import RegressionEvaluator

# Create data subsets at 20%, 50%, 75%, and 100% of the full dataset
fractions = {"20%": 0.2, "50%": 0.5, "75%": 0.75, "100%": 1.0}
data_subsets = {}

# Generate subsets and store them in a dictionary
for key, fraction in fractions.items():
    data_subsets[key] = df_cleaned.sample(fraction=fraction, seed=42)
```

```
print("Data subsets created for 20%, 50%, 75%, and 100%")
```

Data subsets created for 20%, 50%, 75%, and 100%

[63]:
```
# Initialize models
rf = RandomForestRegressor(featuresCol="features",␣
↪labelCol="total_response_time", predictionCol="prediction", numTrees=100)
gbt = GBTRegressor(featuresCol="features", labelCol="total_response_time",␣
↪predictionCol="prediction", maxIter=100)

# Initialize evaluator
evaluator_rmse = RegressionEvaluator(labelCol="total_response_time",␣
↪predictionCol="prediction", metricName="rmse")
evaluator_r2 = RegressionEvaluator(labelCol="total_response_time",␣
↪predictionCol="prediction", metricName="r2")

# Results storage
results = []

# Train and evaluate both models on each subset
for size, df_subset in data_subsets.items():
    # Assemble features
    assembler = VectorAssembler(inputCols=["dispatch_time", "arrival_time"],␣
↪outputCol="features")
    df_prepared = assembler.transform(df_subset).select("features",␣
↪"total_response_time")

    # Split data
    train_data, test_data = df_prepared.randomSplit([0.8, 0.2], seed=42)

    # Random Forest
    start_time = time.time()
    rf_model = rf.fit(train_data)
    rf_predictions = rf_model.transform(test_data)
    rf_time = time.time() - start_time
    rf_rmse = evaluator_rmse.evaluate(rf_predictions)
    rf_r2 = evaluator_r2.evaluate(rf_predictions)

    # Gradient Boosting
    start_time = time.time()
    gbt_model = gbt.fit(train_data)
    gbt_predictions = gbt_model.transform(test_data)
    gbt_time = time.time() - start_time
    gbt_rmse = evaluator_rmse.evaluate(gbt_predictions)
    gbt_r2 = evaluator_r2.evaluate(gbt_predictions)
```

```
    # Store results
    results.append((size, rf_time, rf_rmse, rf_r2, gbt_time, gbt_rmse, gbt_r2))
    print(f"Data Size: {size} - RF: Time={rf_time:.2f}s, RMSE={rf_rmse:.4f},␣
↪R²={rf_r2:.4f} | GBT: Time={gbt_time:.2f}s, RMSE={gbt_rmse:.4f}, R²={gbt_r2:.
↪4f}")
```

```
Data Size: 20% - RF: Time=227.44s, RMSE=3.6436, R²=0.9763 | GBT: Time=290.58s,
RMSE=1.6169, R²=0.9953

Data Size: 50% - RF: Time=264.25s, RMSE=3.5704, R²=0.9771 | GBT: Time=362.53s,
RMSE=1.6424, R²=0.9952

Data Size: 75% - RF: Time=290.90s, RMSE=3.5671, R²=0.9771 | GBT: Time=415.53s,
RMSE=1.6325, R²=0.9952

[Stage 5328:>                                                          (0 + 1) / 1]

Data Size: 100% - RF: Time=321.87s, RMSE=3.6491, R²=0.9761 | GBT: Time=473.94s,
RMSE=1.6164, R²=0.9953
```

[64]:
```
import pandas as pd

# Convert results to DataFrame
results_df = pd.DataFrame(results, columns=["Data Size",
                                            "RF Training Time", "RF RMSE", "RF␣
↪R²",
                                            "GBT Training Time", "GBT RMSE",␣
↪"GBT R²"])

# Display results
print("Scale-In Experiment Results:")
print(results_df)
```

```
Scale-In Experiment Results:
  Data Size  RF Training Time   RF RMSE     RF R²  GBT Training Time  \
0       20%        227.437624  3.643566  0.976288         290.576475
1       50%        264.249860  3.570368  0.977129         362.531611
2       75%        290.900296  3.567099  0.977091         415.527479
3      100%        321.868116  3.649114  0.976130         473.944690

   GBT RMSE    GBT R²
0  1.616928  0.995330
1  1.642350  0.995161
```

```
2  1.632526  0.995202
3  1.616351  0.995317
```

[65]:
```
from tabulate import tabulate

# Display the table with borders
print(tabulate(results_df, headers="keys", tablefmt="grid"))
```

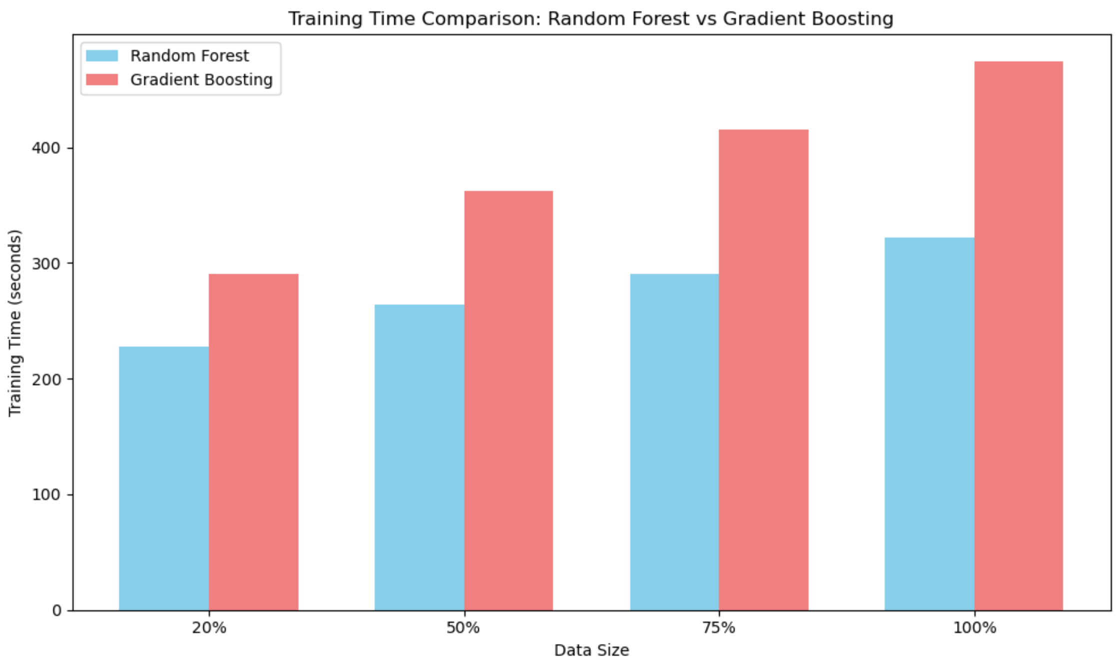
```
+----+-------------+-------------------+----------+---------+---------------
-----+------------+----------+
|    | Data Size   |   RF Training Time |   RF RMSE |    RF R² |   GBT Training
Time |   GBT RMSE |   GBT R² |
+====+=============+===================+==========+=========+===============
=====+============+==========+
|  0 | 20%         |           227.438 |   3.64357 | 0.976288 |
290.576 |    1.61693 | 0.99533  |
+----+-------------+-------------------+----------+---------+---------------
-----+------------+----------+
|  1 | 50%         |           264.25  |   3.57037 | 0.977129 |
362.532 |    1.64235 | 0.995161 |
+----+-------------+-------------------+----------+---------+---------------
-----+------------+----------+
|  2 | 75%         |           290.9   |   3.5671  | 0.977091 |
415.527 |    1.63253 | 0.995202 |
+----+-------------+-------------------+----------+---------+---------------
-----+------------+----------+
|  3 | 100%        |           321.868 |   3.64911 | 0.97613  |
473.945 |    1.61635 | 0.995317 |
+----+-------------+-------------------+----------+---------+---------------
-----+------------+----------+
```

[66]:
```
from IPython.display import display, HTML

# Render DataFrame as an HTML table with borders
html_table = results_df.to_html(index=False, border=1, justify="center")
display(HTML(html_table))
```

```
<IPython.core.display.HTML object>
```

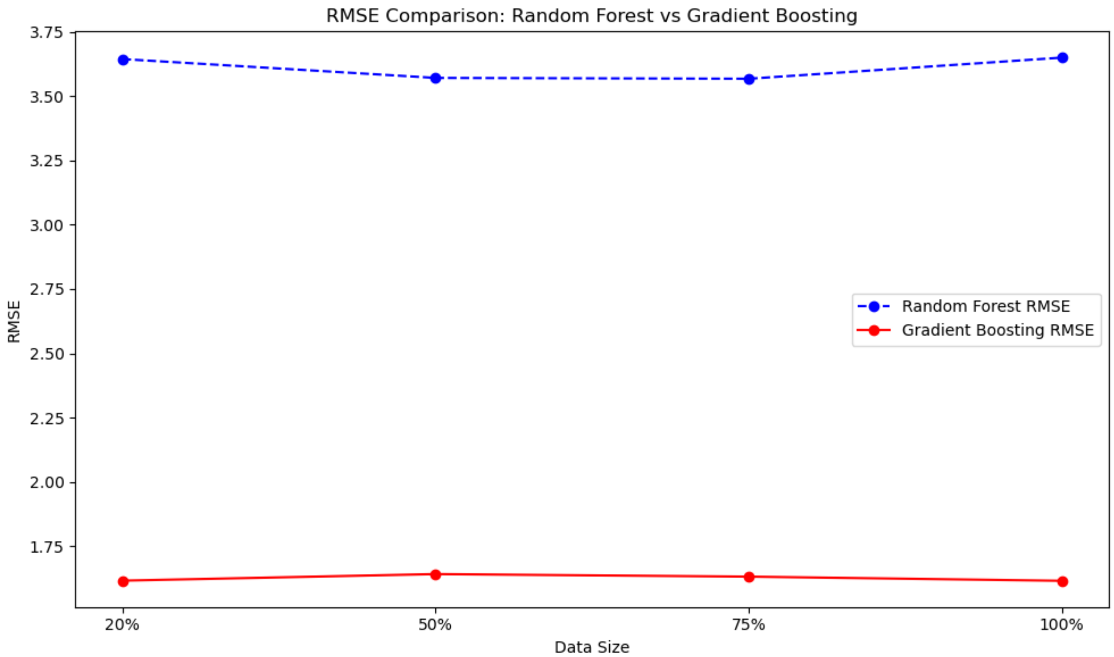
[67]:
```
import matplotlib.pyplot as plt
import numpy as np

# Data for plotting
data_sizes = results_df["Data Size"]
rf_times = results_df["RF Training Time"]
gbt_times = results_df["GBT Training Time"]

# Bar width
```

```
bar_width = 0.35
index = np.arange(len(data_sizes))

# Plot Training Time
plt.figure(figsize=(10, 6))
plt.bar(index, rf_times, bar_width, label="Random Forest", color="skyblue")
plt.bar(index + bar_width, gbt_times, bar_width, label="Gradient Boosting",␣
 ↪color="lightcoral")

# Add labels and title
plt.xlabel("Data Size")
plt.ylabel("Training Time (seconds)")
plt.title("Training Time Comparison: Random Forest vs Gradient Boosting")
plt.xticks(index + bar_width / 2, data_sizes)
plt.legend()

# Display the plot
plt.tight_layout()
plt.show()
```



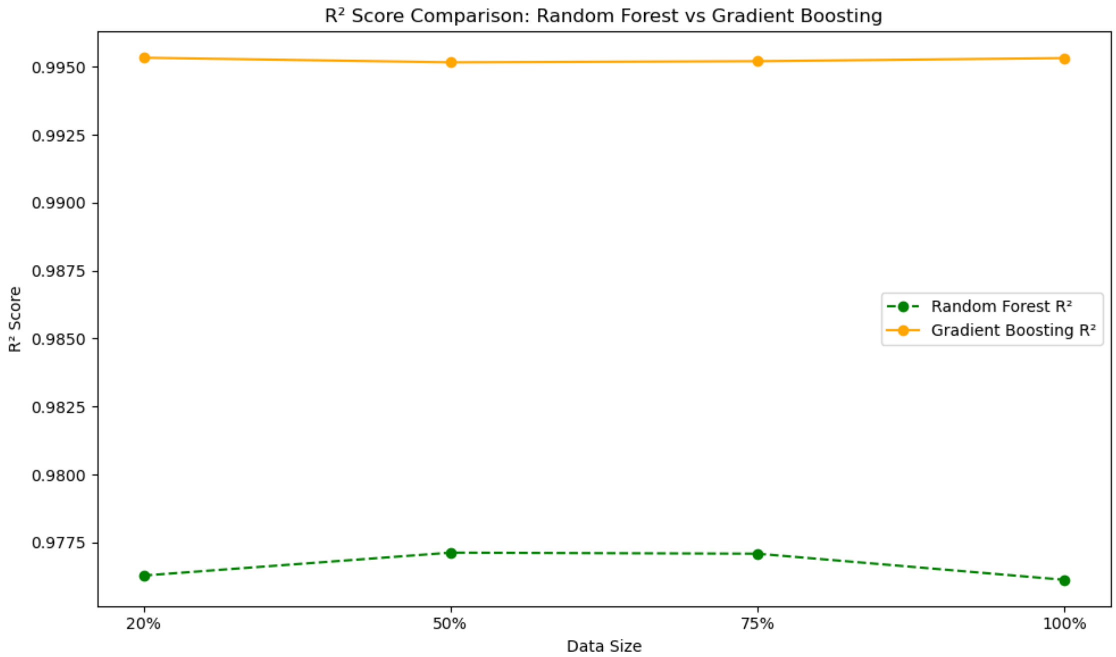
[68]:
```
# Data for plotting
rf_rmse = results_df["RF RMSE"]
gbt_rmse = results_df["GBT RMSE"]

# Plot RMSE Comparison
```

```
plt.figure(figsize=(10, 6))
plt.plot(data_sizes, rf_rmse, marker="o", label="Random Forest RMSE",␣
 ↪color="blue", linestyle="--")
plt.plot(data_sizes, gbt_rmse, marker="o", label="Gradient Boosting RMSE",␣
 ↪color="red", linestyle="-")

# Add labels and title
plt.xlabel("Data Size")
plt.ylabel("RMSE")
plt.title("RMSE Comparison: Random Forest vs Gradient Boosting")
plt.legend()

# Display the plot
plt.tight_layout()
plt.show()
```



[69]:
```
# Data for plotting
rf_r2 = results_df["RF R²"]
gbt_r2 = results_df["GBT R²"]

# Plot R² Comparison
plt.figure(figsize=(10, 6))
plt.plot(data_sizes, rf_r2, marker="o", label="Random Forest R²",␣
 ↪color="green", linestyle="--")
```

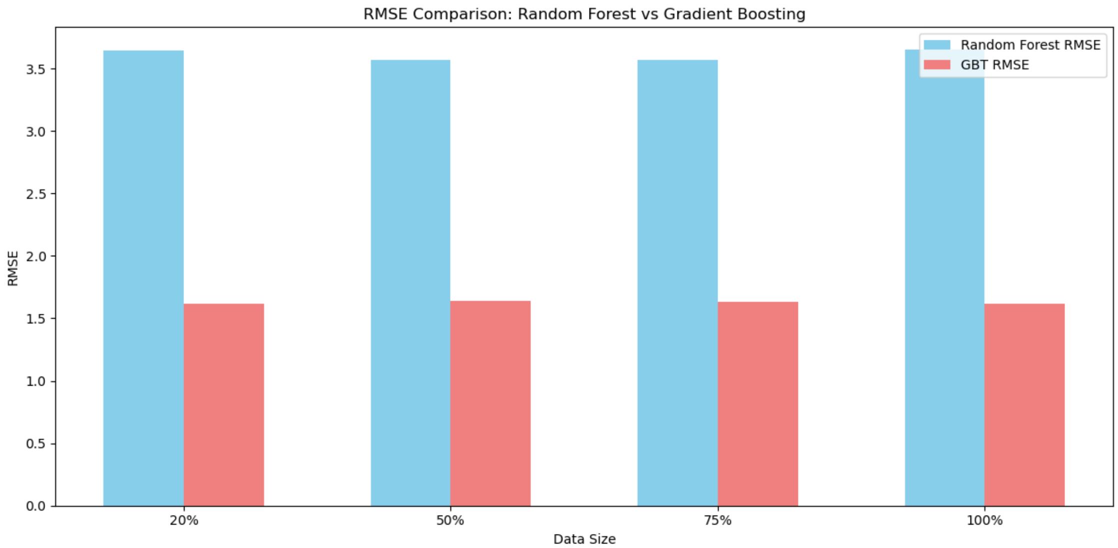
```
plt.plot(data_sizes, gbt_r2, marker="o", label="Gradient Boosting R²",
 ↪color="orange", linestyle="-")

# Add labels and title
plt.xlabel("Data Size")
plt.ylabel("R² Score")
plt.title("R² Score Comparison: Random Forest vs Gradient Boosting")
plt.legend()

# Display the plot
plt.tight_layout()
plt.show()
```

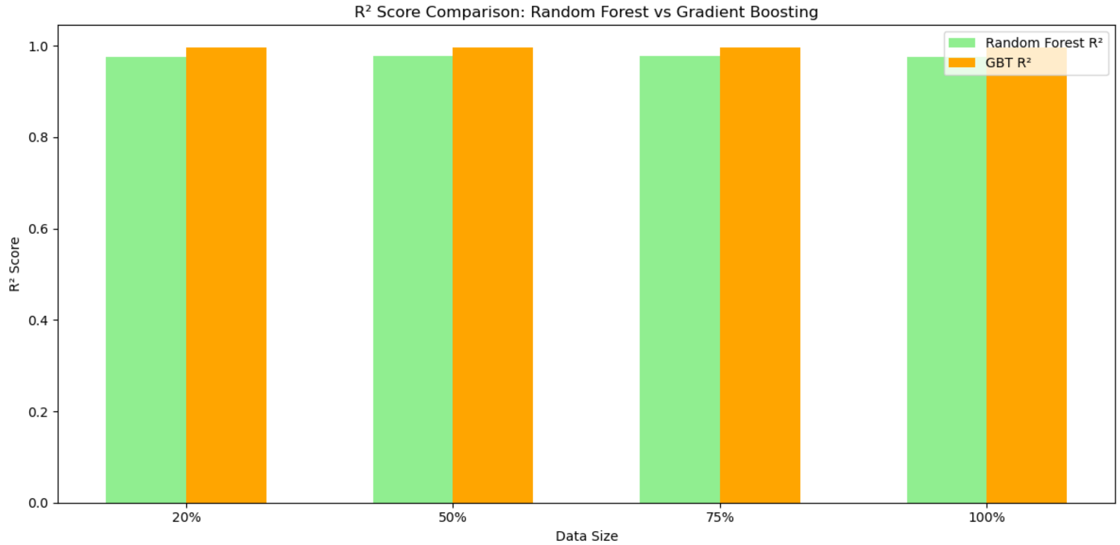


[70]:

```
# Data for plotting
data_sizes = results_df["Data Size"]
rf_rmse = results_df["RF RMSE"]
gbt_rmse = results_df["GBT RMSE"]
rf_r2 = results_df["RF R²"]
gbt_r2 = results_df["GBT R²"]

# Bar width and positions
bar_width = 0.3
index = np.arange(len(data_sizes))

# Plot RMSE
```

```
plt.figure(figsize=(12, 6))
plt.bar(index - bar_width/2, rf_rmse, bar_width, label="Random Forest RMSE",␣
↪color="skyblue")
plt.bar(index + bar_width/2, gbt_rmse, bar_width, label="GBT RMSE",␣
↪color="lightcoral")

plt.xlabel("Data Size")
plt.ylabel("RMSE")
plt.title("RMSE Comparison: Random Forest vs Gradient Boosting")
plt.xticks(index, data_sizes)
plt.legend()
plt.tight_layout()
plt.show()

# Plot R²
plt.figure(figsize=(12, 6))
plt.bar(index - bar_width/2, rf_r2, bar_width, label="Random Forest R²",␣
↪color="lightgreen")
plt.bar(index + bar_width/2, gbt_r2, bar_width, label="GBT R²", color="orange")

plt.xlabel("Data Size")
plt.ylabel("R² Score")
plt.title("R² Score Comparison: Random Forest vs Gradient Boosting")
plt.xticks(index, data_sizes)
plt.legend()
plt.tight_layout()
plt.show()
```




R² Score Comparison: Random Forest vs Gradient Boosting
Random Forest R²
GBT R²
R² Score
1.0
0.8
0.6
0.4
0.2
0.0
20%
50%
75%
100%
Data Size

[ ]: