databrickslatest

(https://databricks.com)

```
import pandas as pd
import pyspark.sql.functions as F
import seaborn as sns
import matplotlib.pyplot as plt
from pyspark.sql import SparkSession
from pyspark.sql.utils import AnalysisException
import pyspark.sql.types as T
import os
```

```
spark = SparkSession.builder.getOrCreate()
DIRECTORY = "dbfs:/FileStore/tables"
logs_1 = spark.read.csv(
    path=os.path.join(DIRECTORY, "NYPD_Calls_for_Service__Year_to_Date_.csv"),
    sep=",",
    header=True,
    inferSchema=True,
    timestampFormat="mm-dd-yyyy",
  )
  logs_1.count()
  logs_1.printSchema()
```

```
root
 |-- CAD_EVNT_ID: integer (nullable = true)
 |-- CREATE_DATE: date (nullable = true)
 |-- INCIDENT_DATE: date (nullable = true)
 |-- INCIDENT_TIME: string (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)
```

DataFrame logs_1 by various columns, such as 'Latitude,' 'Longitude,' 'BORO_NM' (borough name), 'TYP_DESC' (incident type), 'NYPD_PCT_CD' (NYPD precinct code), 'PATRL_BORO_NM' (patrol borough name), 'RADIO_CODE,' and 'ARRIVD_TS' (arrival timestamp). Calculate the count of occurrences for each unique value in these columns, order the results in descending order based on the count, and display the summarized information. Analyses provide insights into the distribution and frequency of incident data across different attributes.

```
logs_1.groupBy(F.col("Latitude")).count().orderBy(F.col("count").desc()).show()
```

```
+---------+-----+
| Latitude|count|
+---------+-----+
|40.756642|27157|
|40.840849| 9339|
|40.804046| 9192|
|40.747051| 8987|
|  40.7516| 8662|
|40.752734| 8085|
|40.576285| 7580|
|40.794953| 7207|
|40.668884| 7158|
|40.808374| 7062|
|40.734961| 6715|
|40.827825| 6588|
|40.749218| 6570|
|40.684462| 6560|
|40.811113| 6150|
|  40.6704| 6072|
|40.732019| 5954|
|40.842273| 5812|
```

```
logs_1.groupBy(F.col("Longitude")).count().orderBy(F.col("count").desc()).show()
```

```
+----------+-----+
| Longitude|count|
+----------+-----+
|-73.988372|27178|
|-73.940055| 9497|
| -73.93662| 9192|
|-73.891472| 8992|
|-73.992043| 8662|
|-73.976548| 8078|
|-73.981391| 7580|
|-73.971437| 7207|
|-73.932506| 7091|
```

```
|-73.946886| 6877|
| -73.99049| 6715|
|-73.925345| 6689|
|-73.868944| 6570|
|-73.977751| 6560|
|-73.952349| 6173|
|-73.956624| 6072|
|-74.000734| 5951|
```

```
logs_1.groupBy(F.col("BORO_NM")).count().orderBy(F.col("count").desc()).show()
```

```
+-------------+-------+
|      BORO_NM|  count|
+-------------+-------+
|     BROOKLYN|1071759|
|    MANHATTAN|1047818|
|        BRONX| 709771|
|       QUEENS| 673645|
|STATEN ISLAND| 134421|
|       (null)|     72|
+-------------+-------+
```

```
logs_1.groupBy(F.col("TYP_DESC")).count().orderBy(F.col("count").desc()).show()
```

```
+--------------------+------+
|           TYP_DESC| count|
+--------------------+------+
|VISIBILITY PATROL...|459220|
|STATION INSPECTIO...|335174|
|SEE COMPLAINANT: ...|275269|
|TRANSIT PATROL/IN...|172149|
|TRAIN RUN/MOBILE ...|166117|
|INVESTIGATE/POSSI...|123919|
|AMBULANCE CASE: E...|121765|
|VISIBILITY PATROL...|109511|
|INVESTIGATE/POSSI...|104356|
|INVESTIGATE/POSSI...| 87669|
|ALARMS: COMMERCIA...| 73058|
|     DISPUTE: INSIDE| 66832|
|     DISPUTE: FAMILY| 62456|
|VEHICLE ACCIDENT:...| 61045|
|VISIBILITY PATROL...| 52642|
|      COMMUNITY TIME| 51455|
|VISIBILITY PATROL...| 47900|
|      TRAFFIC SAFETY| 46631|
```

```
logs_1.groupBy(F.col("NYPD_PCT_CD")).count().orderBy(F.col("count").desc()).show()
```

```
+-----------+------+
|NYPD_PCT_CD| count|
+-----------+------+
|         14|109252|
|         75| 91122|
|         40| 90337|
|         44| 79523|
|         52| 78372|
|         73| 75756|
|         43| 71203|
|         84| 70859|
|         18| 67400|
|         47| 66581|
|         13| 64047|
|        115| 62247|
|         24| 58700|
|        114| 58625|
|          1| 58460|
|         46| 56127|
|        108| 56067|
|         25| 55561|
```

```
logs_1.groupBy(F.col("PATRL_BORO_NM")).count().orderBy(F.col("count").desc()).show()
```

```
+--------------------+------+
|       PATRL_BORO_NM| count|
+--------------------+------+
|   PATROL BORO BRONX|709771|
|PATROL BORO BKLYN...|537463|
|PATROL BORO MAN S...|535314|
|PATROL BORO BKLYN...|534296|
|PATROL BORO MAN N...|512504|
|PATROL BORO QUEEN...|366262|
|PATROL BORO QUEEN...|307383|
|PATROL BORO STATE...|134421|
|              (null)|    72|
+--------------------+------+
```

```
logs_1.groupBy(F.col("RADIO_CODE")).count().orderBy(F.col("count").desc()).show()
```

```
+----------+------+
|RADIO_CODE| count|
+----------+------+
|       75D|459220|
```

```
|       75S|335174|
|      68Q1|275269|
|       75T|172149|
|       75M|166117|
|      10H1|123919|
|      54E1|121765|
|       75F|109511|
|      10Y3|104356|
|      10V2| 87669|
|      11C4| 73058|
|      52D1| 66832|
|      52D6| 62456|
|       53S| 61045|
|       75P| 52642|
|       75C| 51455|
|       75I| 47900|
|       75Z| 46631|
```

```
logs_1.groupBy(F.col("ARRIVD_TS")).count().orderBy(F.col("count").desc()).show()
```

```
+-------------------+------+
|          ARRIVD_TS| count|
+-------------------+------+
|               null|735098|
|05/09/2023 10:07:...|     7|
|06/28/2023 09:20:...|     7|
|02/04/2023 05:45:...|     6|
|02/02/2023 01:24:...|     5|
|01/19/2023 04:44:...|     5|
|04/29/2023 01:41:...|     5|
|01/01/2023 04:51:...|     5|
|05/19/2023 04:36:...|     5|
|01/10/2023 05:38:...|     5|
|06/06/2023 09:39:...|     5|
|01/20/2023 05:11:...|     5|
|05/19/2023 05:01:...|     5|
|01/07/2023 04:19:...|     5|
|06/19/2023 02:33:...|     5|
|03/14/2023 06:51:...|     5|
|03/02/2023 05:01:...|     5|
|03/03/2023 09:12:...|     5|
```

Removes specific columns, including 'RADIO_CODE,' 'Latitude,' 'Longitude,' and 'ARRIVD_TS,' from the DataFrame logs_1

```
  #Dropping Latitudes, Longitudes and RADIO_CODE
  logs_1 = logs_1.drop("RADIO_CODE","Latitude","Longitude")
  #Dropping ARRIVD_TS
  logs_1 = logs_1.drop("ARRIVD_TS")


  logs_1.select(F.col("ADD_TS")).show(5)
  print(logs_1.select(F.col("ADD_TS")).dtypes)
```

```
+--------------------+
|              ADD_TS|
+--------------------+
|01/01/2023 01:08:...|
|01/01/2023 12:38:...|
|01/01/2023 12:01:...|
|01/01/2023 12:01:...|
|01/01/2023 12:01:...|
+--------------------+
only showing top 5 rows

[('ADD_TS', 'string')]
```

Extract and format various date and time components from the columns 'ADD_TS,' 'DISP_TS,' 'CLOSNG_TS,' 'CREATE_DATE,' and 'INCIDENT_DATE' in the DataFrame logs_1. The formatted components include month, day, year, hour, minutes, and seconds.

```
  logs_1.select(
      F.col("ADD_TS"),
      F.col("ADD_TS").substr(1,2).cast("int").alias("ADD_TS_Month"),
      F.col("ADD_TS").substr(4,2).cast("int").alias("ADD_TS_Day"),
      F.col("ADD_TS").substr(7,4).cast("int").alias("ADD_TS_Year"),
      F.col("ADD_TS").substr(12,2).cast("int").alias("ADD_TS_Hour"),
      F.col("ADD_TS").substr(15,2).cast("int").alias("ADD_TS_Minutes"),
      F.col("ADD_TS").substr(18,2).cast("int").alias("ADD_TS_Seconds"),
  ).distinct().show(5)
```

```
+--------------------+------------+----------+-----------+-----------+-------------+-------------+
|              ADD_TS|ADD_TS_Month|ADD_TS_Day|ADD_TS_Year|ADD_TS_Hour|ADD_TS_Minutes|ADD_TS_Seconds|
+--------------------+------------+----------+-----------+-----------+-------------+-------------+
|01/01/2023 12:01:...|           1|         1|       2023|         12|            1|           29|
|01/01/2023 12:01:...|           1|         1|       2023|         12|            1|           34|
|01/01/2023 12:01:...|           1|         1|       2023|         12|            1|           26|
|01/01/2023 12:38:...|           1|         1|       2023|         12|           38|            0|
|01/01/2023 12:01:...|           1|         1|       2023|         12|            1|           35|
+--------------------+------------+----------+-----------+-----------+-------------+-------------+
```

```
only showing top 5 rows
```

```
  logs_1.select(
      F.col("DISP_TS"),
      F.col("DISP_TS").substr(1,2).cast("int").alias("DISP_TS_Month"),
      F.col("DISP_TS").substr(4,2).cast("int").alias("DISP_TS_Day"),
      F.col("DISP_TS").substr(7,4).cast("int").alias("DISP_TS_Year"),
      F.col("DISP_TS").substr(12,2).cast("int").alias("DISP_TS_Hour"),
      F.col("DISP_TS").substr(15,2).cast("int").alias("DISP_TS_Minutes"),
      F.col("DISP_TS").substr(18,2).cast("int").alias("DSIP_TS_Seconds"),
  ).distinct().show(5)
```

```
+-----------------+-------------+-----------+------------+------------+---------------+---------------+
|          DISP_TS|DISP_TS_Month|DISP_TS_Day|DISP_TS_Year|DISP_TS_Hour|DISP_TS_Minutes|DSIP_TS_Seconds|
+-----------------+-------------+-----------+------------+------------+---------------+---------------+
|01/01/2023 12:37:...|            1|          1|        2023|          12|             37|             14|
|01/01/2023 12:06:...|            1|          1|        2023|          12|              6|             18|
|01/01/2023 02:40:...|            1|          1|        2023|           2|             40|             24|
|01/01/2023 12:38:...|            1|          1|        2023|          12|             38|             34|
|01/01/2023 01:09:...|            1|          1|        2023|           1|              9|             57|
+-----------------+-------------+-----------+------------+------------+---------------+---------------+
only showing top 5 rows
```

```
  logs_1.select(
      F.col("CLOSNG_TS"),
      F.col("CLOSNG_TS").substr(1,2).cast("int").alias("CLOSNG_TS_Month"),
      F.col("CLOSNG_TS").substr(4,2).cast("int").alias("CLOSNG_TS_Day"),
      F.col("CLOSNG_TS").substr(7,4).cast("int").alias("CLOSNG_TS_Year"),
      F.col("CLOSNG_TS").substr(12,2).cast("int").alias("CLOSNG_TS_Hour"),
      F.col("CLOSNG_TS").substr(15,2).cast("int").alias("CLOSNG_TS_Minutes"),
      F.col("CLOSNG_TS").substr(18,2).cast("int").alias("CLOSNG_TS_Seconds"),
  ).distinct().show(5)
```

```
+-----------------+--------------+------------+-------------+-------------+-----------------+-----------------+
|          CLOSNG_TS|CLOSNG_TS_Month|CLOSNG_TS_Day|CLOSNG_TS_Year|CLOSNG_TS_Hour|CLOSNG_TS_Minutes|CLOSNG_TS_Seconds|
+-----------------+--------------+------------+-------------+-------------+-----------------+-----------------+
|01/01/2023 12:06:...|             1|           1|         2023|           12|                6|               27|
|01/01/2023 01:57:...|             1|           1|         2023|            1|               57|               44|
|01/01/2023 01:24:...|             1|           1|         2023|            1|               24|               22|
|01/01/2023 01:45:...|             1|           1|         2023|            1|               45|               21|
|01/01/2023 02:26:...|             1|           1|         2023|            2|               26|               19|
+-----------------+--------------+------------+-------------+-------------+-----------------+-----------------+
```

```
only showing top 5 rows
```

```
  logs_1.select(
      F.col("INCIDENT_DATE"),
      F.col("INCIDENT_DATE").substr(1,4).cast("int").alias("INCIDENT_DATE_Year"),
      F.col("INCIDENT_DATE").substr(6,2).cast("int").alias("INCIDENT_DATE_Month"),
      F.col("INCIDENT_DATE").substr(9,2).cast("int").alias("INCIDENT_DATE_Day"),
  ).distinct().show(5)
```

```
+-------------+------------------+-------------------+-----------------+
|INCIDENT_DATE|INCIDENT_DATE_Year|INCIDENT_DATE_Month|INCIDENT_DATE_Day|
+-------------+------------------+-------------------+-----------------+
|   2023-01-04|              2023|                  1|                4|
|   2023-01-03|              2023|                  1|                3|
|   2023-01-05|              2023|                  1|                5|
|   2022-12-31|              2022|                 12|               31|
|   2023-01-01|              2023|                  1|                1|
+-------------+------------------+-------------------+-----------------+
only showing top 5 rows
```

Group the data based on the 'ADD_TS,' 'DISP_TS,' and 'CLOSNG_TS' columns, counting the occurrences for each timestamp and ordering the results in descending order by count.

```
  logs_1.groupBy(F.col("ADD_TS")).count().orderBy(F.col("count").desc()).show()
```

```
+--------------------+-----+
|              ADD_TS|count|
+--------------------+-----+
|05/10/2023 02:38:...|    6|
|03/23/2023 04:55:...|    6|
|04/12/2023 11:12:...|    6|
|04/09/2023 07:05:...|    6|
|02/20/2023 12:29:...|    6|
|05/31/2023 09:34:...|    6|
|05/12/2023 11:19:...|    6|
|01/06/2023 11:11:...|    6|
|04/11/2023 04:22:...|    6|
|01/15/2023 08:37:...|    6|
|04/06/2023 07:15:...|    6|
|01/27/2023 03:08:...|    6|
|06/16/2023 11:27:...|    6|
|01/03/2023 09:23:...|    6|
|06/11/2023 02:54:...|    6|
```

```
|03/30/2023 05:14:...|    6|
|01/25/2023 05:02:...|    6|
```

```
logs_1.groupBy(F.col("DISP_TS")).count().orderBy(F.col("count").desc()).show()
```

```
+-------------------+-----+
|            DISP_TS|count|
+-------------------+-----+
|05/21/2023 09:10:...|    6|
|04/28/2023 10:14:...|    6|
|03/07/2023 03:46:...|    6|
|03/02/2023 11:53:...|    6|
|03/07/2023 08:49:...|    6|
|06/22/2023 04:26:...|    5|
|01/18/2023 01:48:...|    5|
|01/02/2023 09:26:...|    5|
|01/21/2023 03:41:...|    5|
|01/14/2023 11:11:...|    5|
|01/21/2023 01:53:...|    5|
|01/30/2023 06:36:...|    5|
|03/03/2023 04:38:...|    5|
|02/02/2023 01:24:...|    5|
|02/02/2023 05:34:...|    5|
|02/04/2023 05:30:...|    5|
|01/07/2023 04:19:...|    5|
|02/16/2023 06:02:...|    5|
```

```
logs_1.groupBy(F.col("CLOSNG_TS")).count().orderBy(F.col("count").desc()).show()
```

```
+-------------------+-----+
|          CLOSNG_TS|count|
+-------------------+-----+
|               null|   14|
|02/13/2023 01:24:...|    9|
|04/28/2023 01:36:...|    9|
|02/28/2023 12:28:...|    8|
|05/22/2023 12:37:...|    8|
|01/20/2023 12:49:...|    7|
|06/01/2023 11:19:...|    7|
|01/09/2023 12:37:...|    7|
|02/14/2023 12:27:...|    7|
|02/10/2023 09:02:...|    7|
|06/30/2023 12:44:...|    7|
|03/21/2023 03:50:...|    7|
|05/13/2023 01:09:...|    7|
|04/16/2023 12:53:...|    7|
```

```
|03/07/2023 05:30:...|    6|
|02/22/2023 04:57:...|    6|
|03/30/2023 08:25:...|    6|
```

Extract the month component from the 'ADD_TS' column, assuming a timestamp format of "MM/dd/yyyy HH:mm:ss." The extracted month is then cast to an integer and stored in a new 'Month' column.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import regexp_extract

# Initialize Spark Session
spark = SparkSession.builder.appName("ExtractMonthWithRegex").getOrCreate()

# Define the regex pattern for the month - assuming the format is "MM/dd/yyyy HH:mm:ss"
month_pattern = '^(\d{2})/'

# Apply regex to extract the month and create a new column
logs_1 = logs_1.withColumn('Month', regexp_extract('ADD_TS', month_pattern, 1))

# Convert the extracted month to integer
logs_1 = logs_1.withColumn('Month', logs_1['Month'].cast('int'))
# logs_1 = logs_1.withColumn('Month', col('Month').cast('int'))

# Display the DataFrame
logs_1.show()
```

```
+-----------+-----------+-------------+-------------+-----------+---------+--------------------+--------+--------+--------------------+------------+--------------------+-------------
-------+--------------------+-----+
|CAD_EVNT_ID|CREATE_DATE|INCIDENT_DATE|INCIDENT_TIME|NYPD_PCT_CD|  BORO_NM|       PATRL_BORO_NM|GEO_CD_X|GEO_CD_Y|           TYP_DESC|    CIP_JOBS|             ADD_TS|
DISP_TS|          CLOSNG_TS|Month|
+-----------+-----------+-------------+-------------+-----------+---------+--------------------+--------+--------+--------------------+------------+--------------------+-------------
-------+--------------------+-----+
|   91250176| 2023-01-01|   2022-12-31|     23:24:39|         67| BROOKLYN|PATROL BORO BKLYN...| 1001878|  175994|VEHICLE ACCIDENT:...|     Non CIP|01/01/2023 01:08:...|01/01/2023 0
1:09:...|01/01/2023 01:57:...|    1|
|   91250180| 2023-01-01|   2022-12-31|     23:24:47|         75| BROOKLYN|PATROL BORO BKLYN...| 1017204|  180778|ALARMS: COMMERCIA...|     Non CIP|01/01/2023 12:38:...|01/01/2023 1
2:38:...|01/01/2023 01:45:...|    1|
|   91250681| 2023-01-01|   2022-12-31|     23:55:56|        114|   QUEENS|PATROL BORO QUEEN...| 1008573|  217117|ALARMS: RESIDENTI...|     Non CIP|01/01/2023 12:01:...|01/01/2023 1
2:06:...|01/01/2023 12:06:...|    1|
|   91250683| 2023-01-01|   2022-12-31|     23:55:59|         66| BROOKLYN|PATROL BORO BKLYN...|  993234|  161780|ALARMS: RESIDENTI...|     Non CIP|01/01/2023 12:01:...|01/01/2023 1
2:37:...|01/01/2023 01:21:...|    1|
|   91250700| 2023-01-01|   2022-12-31|     23:57:08|        115|   QUEENS|PATROL BORO QUEEN...| 1014264|  211852|ALARMS: COMMERCIA...|     Non CIP|01/01/2023 12:01:...|01/01/2023 1
2:14:...|01/01/2023 01:24:...|    1|
|   91250736| 2023-01-01|   2022-12-31|     23:59:09|         46|    BRONX|   PATROL BORO BRONX| 1007356|  248923|ALARMS: COMMERCIA...|     Non CIP|01/01/2023 12:01:...|01/01/2023 0
2:40:...|01/01/2023 02:26:...|    1|
```

```
|   91250746| 2023-01-01|   2023-01-01|     00:00:12|         5|MANHATTAN|PATROL BORO MAN S...|  983903|  200257|SEE COMPLAINANT: ...|     Non CIP|01/01/2023 12:00:...|01/01/2023 1
2:00:...|01/01/2023 02:06:...|    1|
```

Groups the DataFrame 'logs_1' by both the 'Month' and 'TYP_DESC' columns, calculating the count of occurrences for each group. Counts greater than or equal to 5000, and orders the filtered data by 'Month' and count in descending order.

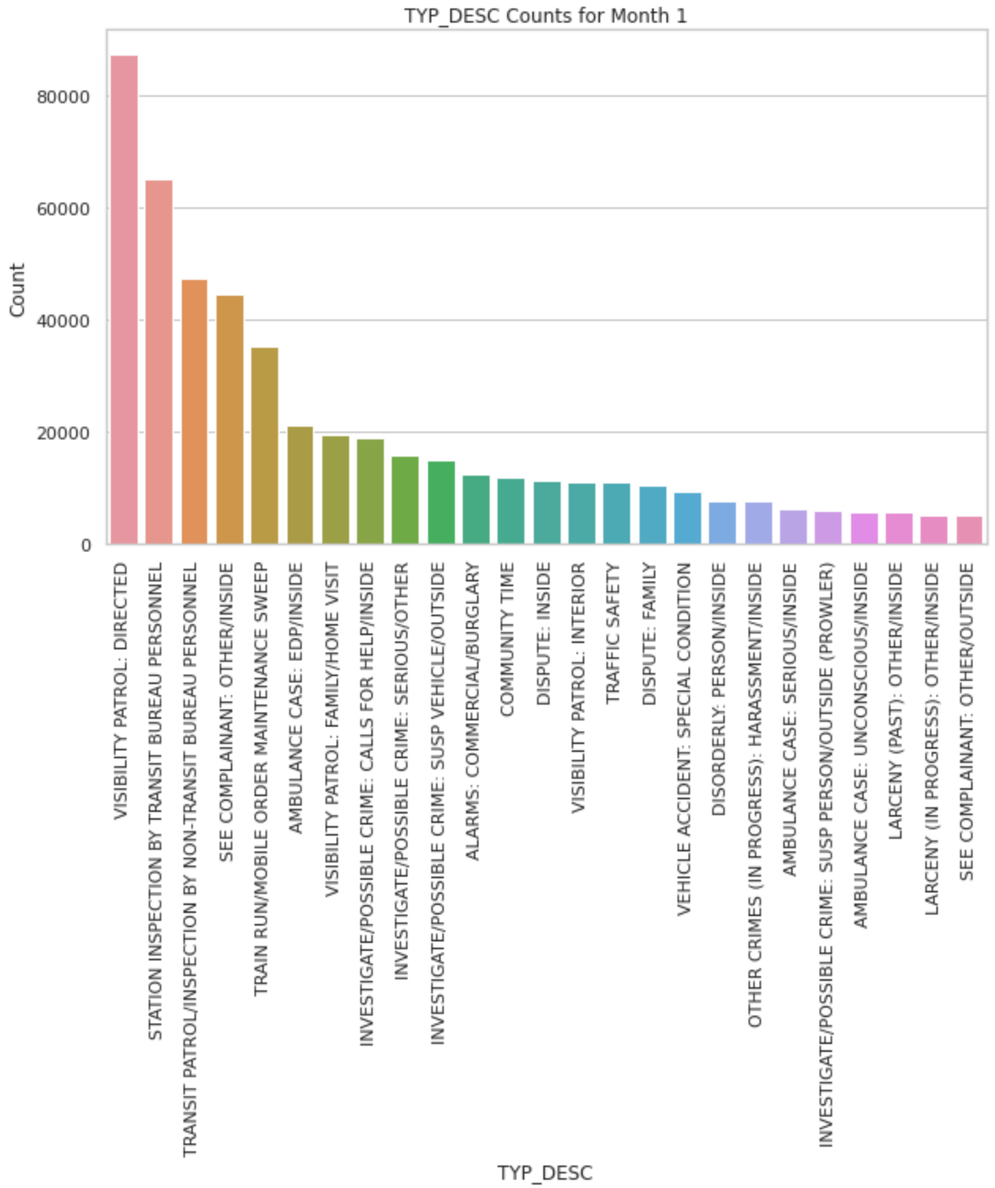
```
grouped_logs_1 = logs_1.groupBy("Month", "TYP_DESC").count()

# Filter to get only TYP_DESC with count >= 5000
filtered_logs_1 = grouped_logs_1.filter(grouped_logs_1['count'] >= 5000)

# Order by Month and count in descending order
ordered_filter_logs_1 = filtered_logs_1.orderBy("Month", F.col("count").desc())

# Show the results
ordered_filter_logs_1.show(truncate=False)
```

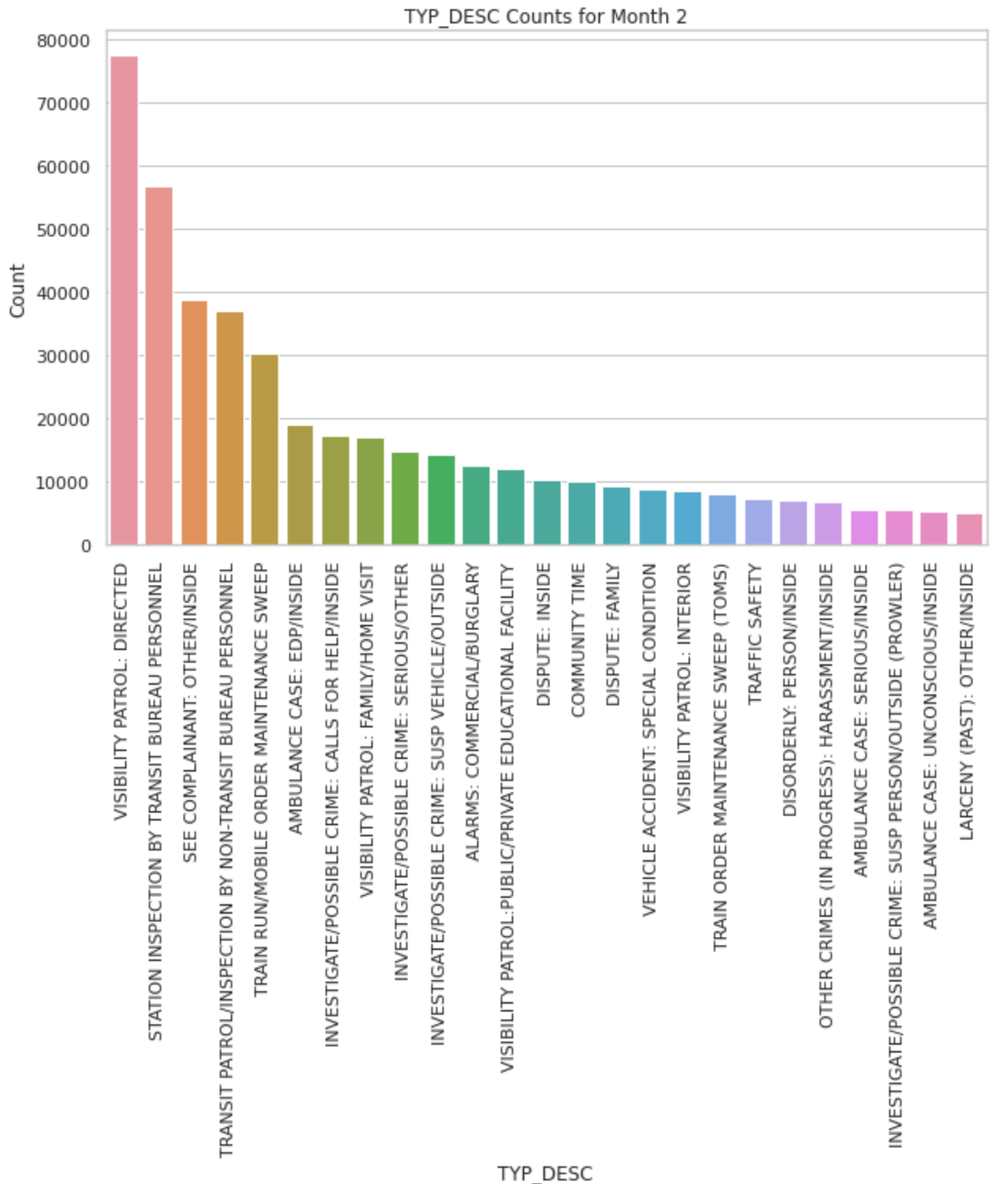
```
+-----+-----------------------------------------------------------+-----+
|Month|TYP_DESC                                                   |count|
+-----+-----------------------------------------------------------+-----+
|1    |VISIBILITY PATROL: DIRECTED                                |87413|
|1    |STATION INSPECTION BY TRANSIT BUREAU PERSONNEL             |65134|
|1    |TRANSIT PATROL/INSPECTION BY NON-TRANSIT BUREAU PERSONNEL|47380|
|1    |SEE COMPLAINANT: OTHER/INSIDE                              |44673|
|1    |TRAIN RUN/MOBILE ORDER MAINTENANCE SWEEP                   |35365|
|1    |AMBULANCE CASE: EDP/INSIDE                                 |21097|
|1    |VISIBILITY PATROL: FAMILY/HOME VISIT                       |19677|
|1    |INVESTIGATE/POSSIBLE CRIME: CALLS FOR HELP/INSIDE          |18956|
|1    |INVESTIGATE/POSSIBLE CRIME: SERIOUS/OTHER                  |15817|
|1    |INVESTIGATE/POSSIBLE CRIME: SUSP VEHICLE/OUTSIDE           |15132|
|1    |ALARMS: COMMERCIAL/BURGLARY                                |12526|
|1    |COMMUNITY TIME                                             |11969|
|1    |DISPUTE: INSIDE                                            |11502|
|1    |VISIBILITY PATROL: INTERIOR                                |11200|
|1    |TRAFFIC SAFETY                                             |11102|
|1    |DISPUTE: FAMILY                                            |10506|
|1    |VEHICLE ACCIDENT: SPECIAL CONDITION                        |9369 |
|1    |DISORDERLY: PERSON/INSIDE                                  |7788 |
```

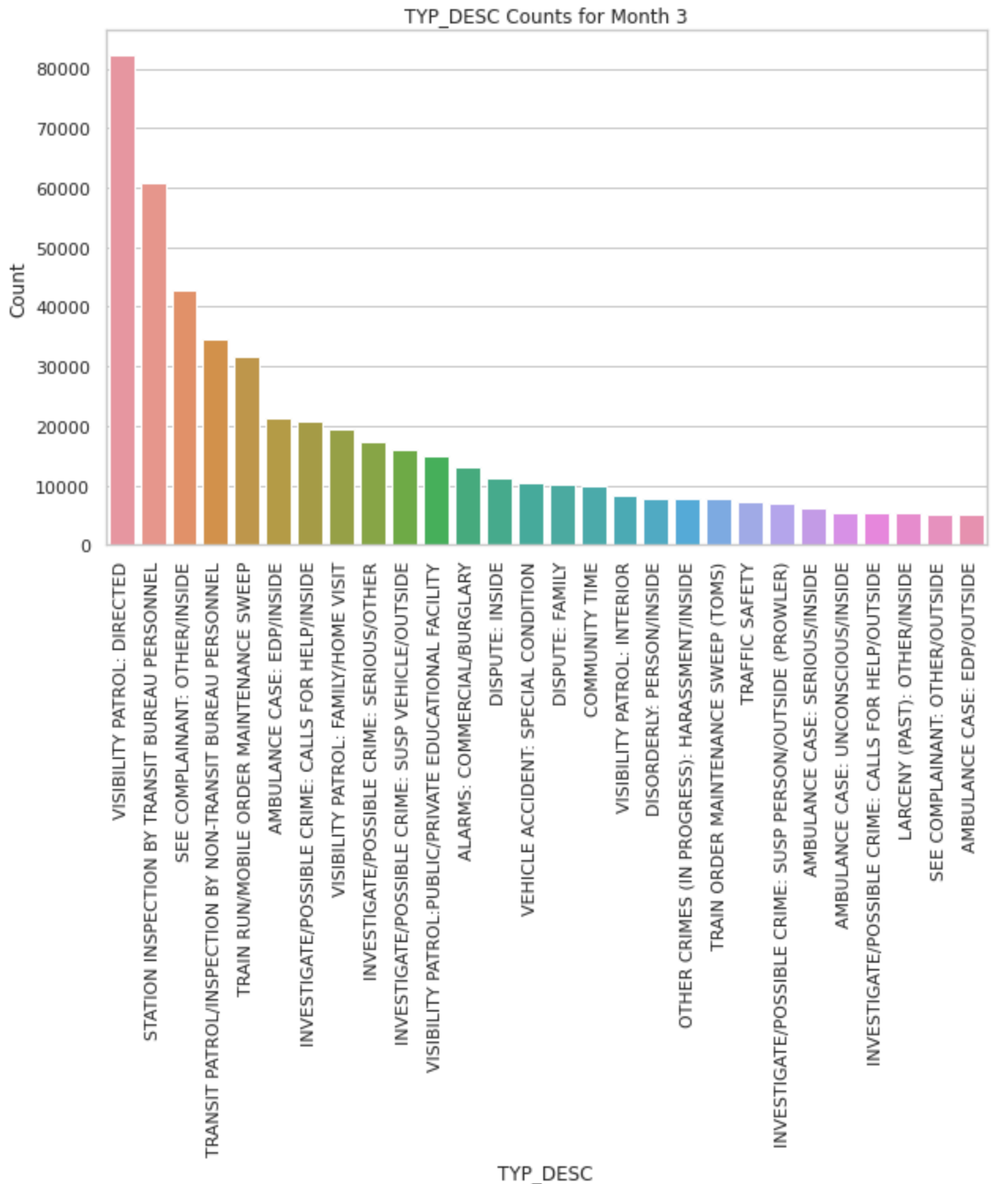
Used the 'matplotlib' and 'seaborn' libraries to create bar plots for each unique month in the DataFrame 'ordered_filter_logs_1.' Filters the data for each specific month, and generates a bar plot displaying the counts of different incident types ('TYP_DESC') for that month.

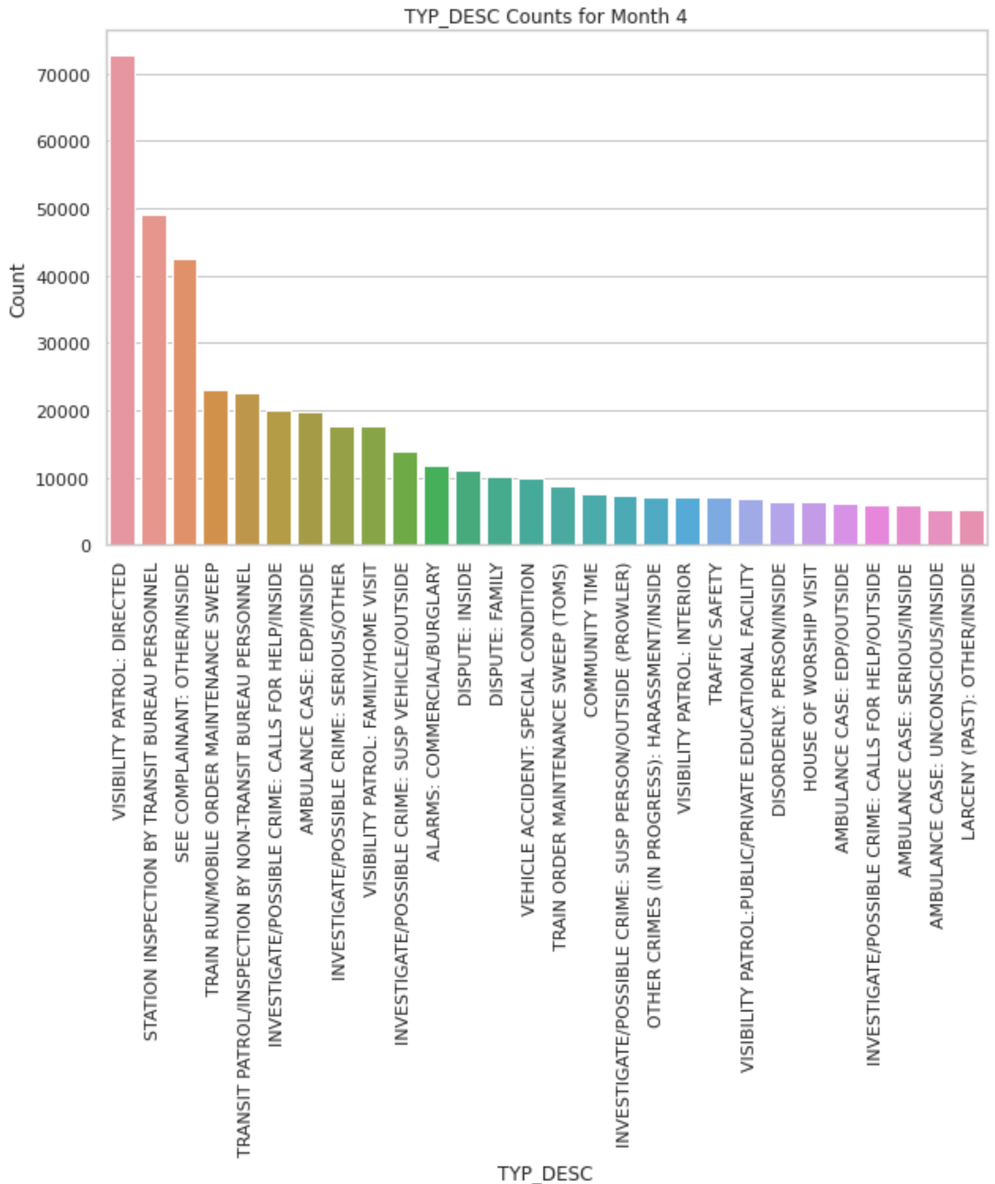
```
!pip install matplotlib seaborn
import matplotlib.pyplot as plt
import seaborn as sns
pandas_df = ordered_filter_logs_1.toPandas()
sns.set(style="whitegrid")

# Get the unique months from the DataFrame
unique_months = pandas_df['Month'].unique()

# Create a plot for each month
for month in unique_months:
    # Filter data for the specific month
    month_data = pandas_df[pandas_df['Month'] == month]

    # Create a bar plot
    plt.figure(figsize=(10, 6))
    sns.barplot(x='TYP_DESC', y='count', data=month_data)

    # Set plot title and labels
    plt.title(f'TYP_DESC Counts for Month {month}')
    plt.xlabel('TYP_DESC')
    plt.ylabel('Count')
    plt.xticks(rotation=90)  # Rotate the x labels for better readability

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

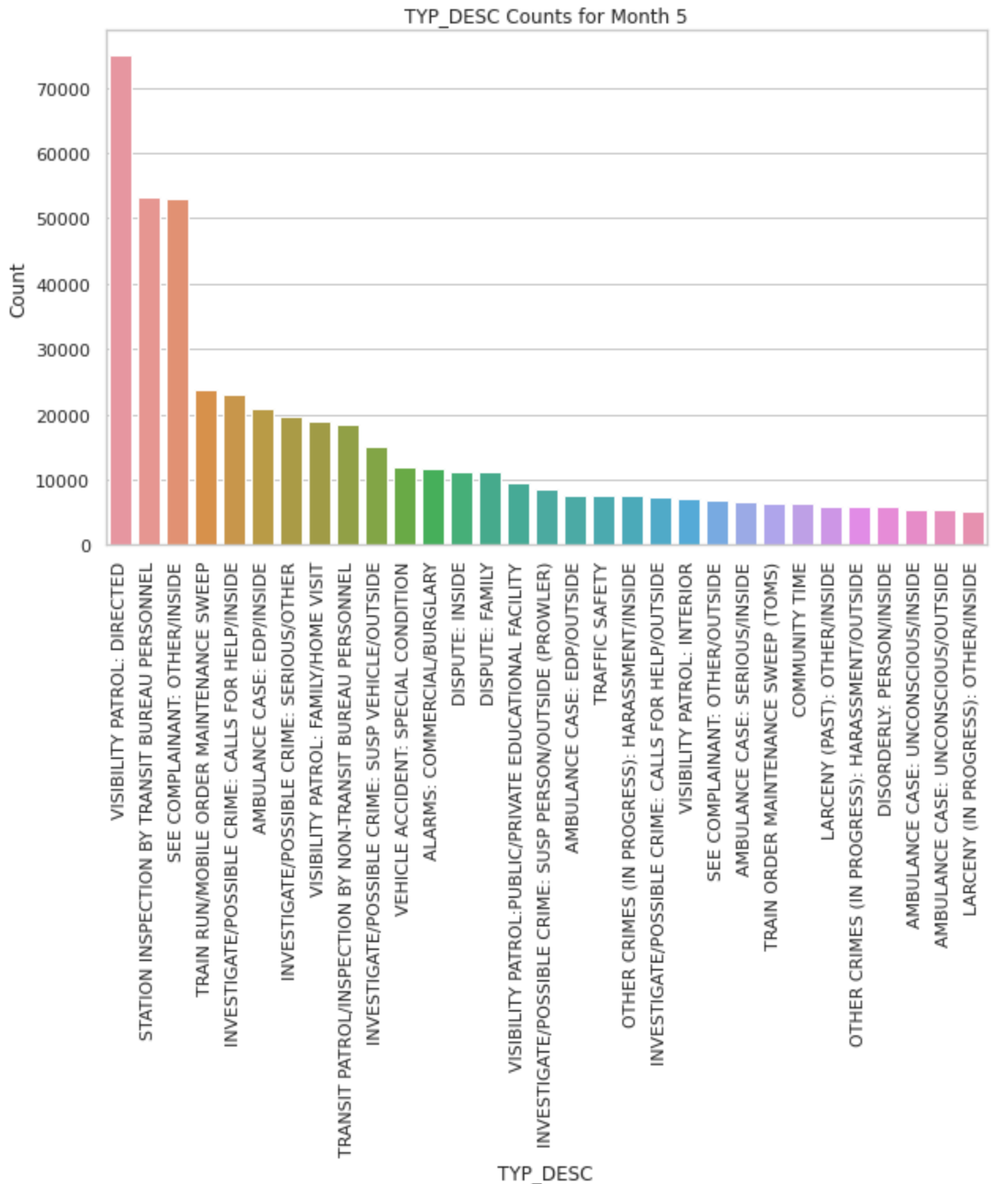
```
Requirement already satisfied: matplotlib in /databricks/python3/lib/python3.9/site-packages (3.5.1)
Requirement already satisfied: seaborn in /databricks/python3/lib/python3.9/site-packages (0.11.2)
Requirement already satisfied: packaging>=20.0 in /databricks/python3/lib/python3.9/site-packages (from matplotlib) (21.3)
Requirement already satisfied: python-dateutil>=2.7 in /databricks/python3/lib/python3.9/site-packages (from matplotlib) (2.8.2)
Requirement already satisfied: fonttools>=4.22.0 in /databricks/python3/lib/python3.9/site-packages (from matplotlib) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /databricks/python3/lib/python3.9/site-packages (from matplotlib) (1.3.2)
Requirement already satisfied: numpy>=1.17 in /databricks/python3/lib/python3.9/site-packages (from matplotlib) (1.21.5)
Requirement already satisfied: pyparsing>=2.2.1 in /databricks/python3/lib/python3.9/site-packages (from matplotlib) (3.0.4)
Requirement already satisfied: cycler>=0.10 in /databricks/python3/lib/python3.9/site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: pillow>=6.2.0 in /databricks/python3/lib/python3.9/site-packages (from matplotlib) (9.0.1)
Requirement already satisfied: scipy>=1.0 in /databricks/python3/lib/python3.9/site-packages (from seaborn) (1.7.3)
Requirement already satisfied: pandas>=0.23 in /databricks/python3/lib/python3.9/site-packages (from seaborn) (1.4.2)
Requirement already satisfied: pytz>=2020.1 in /databricks/python3/lib/python3.9/site-packages (from pandas>=0.23->seaborn) (2021.3)
Requirement already satisfied: six>=1.5 in /databricks/python3/lib/python3.9/site-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
WARNING: You are using pip version 21.2.4; however, version 23.3.2 is available.
You should consider upgrading via the '/local_disk0/.ephemeral_nfs/envs/pythonEnv-d5bfbe7c-8eb6-4808-a53e-4cc9dd6c8df1/bin/python -m pip install --upgrade pip' command.
```

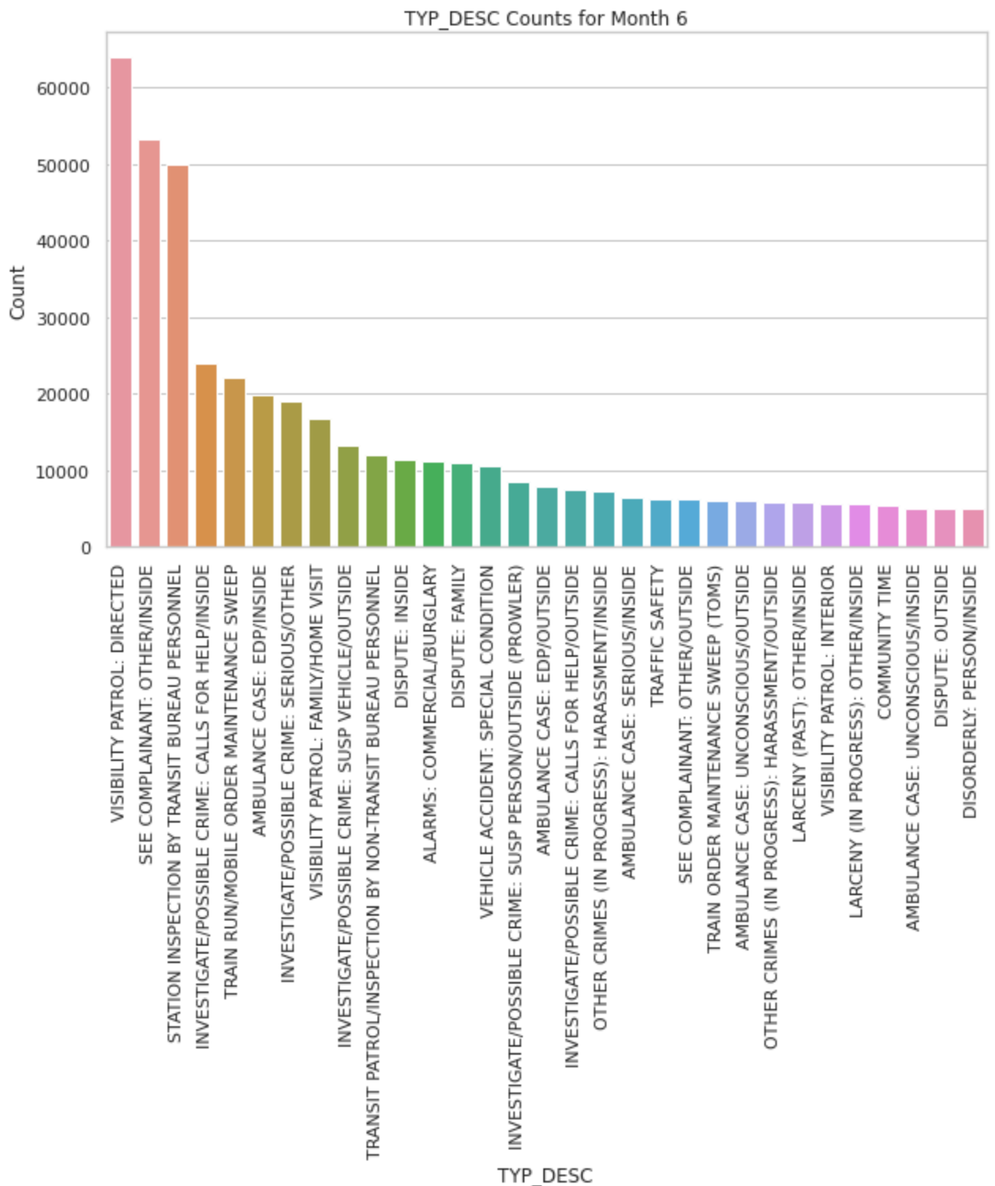

TYP_DESC Counts for Month 1
Count
0
20000
40000
60000
80000
VISIBILITY PATROL: DIRECTED
STATION INSPECTION BY TRANSIT BUREAU PERSONNEL
TRANSIT PATROL/INSPECTION BY NON-TRANSIT BUREAU PERSONNEL
SEE COMPLAINANT: OTHER/INSIDE
TRAIN RUN/MOBILE ORDER MAINTENANCE SWEEP
AMBULANCE CASE: EDP/INSIDE
VISIBILITY PATROL: FAMILY/HOME VISIT
INVESTIGATE/POSSIBLE CRIME: CALLS FOR HELP/INSIDE
INVESTIGATE/POSSIBLE CRIME: SERIOUS/OTHER
INVESTIGATE/POSSIBLE CRIME: SUSP VEHICLE/OUTSIDE
ALARMS: COMMERCIAL/BURGLARY
COMMUNITY TIME
DISPUTE: INSIDE
VISIBILITY PATROL: INTERIOR
TRAFFIC SAFETY
DISPUTE: FAMILY
VEHICLE ACCIDENT: SPECIAL CONDITION
DISORDERLY: PERSON/INSIDE
OTHER CRIMES (IN PROGRESS): HARASSMENT/INSIDE
AMBULANCE CASE: SERIOUS/INSIDE
INVESTIGATE/POSSIBLE CRIME: SUSP PERSON/OUTSIDE (PROWLER)
AMBULANCE CASE: UNCONSCIOUS/INSIDE
LARCENY (PAST): OTHER/INSIDE
LARCENY (IN PROGRESS): OTHER/INSIDE
SEE COMPLAINANT: OTHER/OUTSIDE
TYP_DESC


TYP_DESC Counts for Month 2
Count
0
10000
20000
30000
40000
50000
60000
70000
80000
VISIBILITY PATROL: DIRECTED
STATION INSPECTION BY TRANSIT BUREAU PERSONNEL
SEE COMPLAINANT: OTHER/INSIDE
TRANSIT PATROL/INSPECTION BY NON-TRANSIT BUREAU PERSONNEL
TRAIN RUN/MOBILE ORDER MAINTENANCE SWEEP
AMBULANCE CASE: EDP/INSIDE
INVESTIGATE/POSSIBLE CRIME: CALLS FOR HELP/INSIDE
VISIBILITY PATROL: FAMILY/HOME VISIT
INVESTIGATE/POSSIBLE CRIME: SERIOUS/OTHER
INVESTIGATE/POSSIBLE CRIME: SUSP VEHICLE/OUTSIDE
ALARMS: COMMERCIAL/BURGLARY
VISIBILITY PATROL:PUBLIC/PRIVATE EDUCATIONAL FACILITY
DISPUTE: INSIDE
COMMUNITY TIME
DISPUTE: FAMILY
VEHICLE ACCIDENT: SPECIAL CONDITION
VISIBILITY PATROL: INTERIOR
TRAIN ORDER MAINTENANCE SWEEP (TOMS)
DISORDERLY: TRAFFIC SAFETY
OTHER CRIMES (IN PROGRESS): DISORDERLY: PERSON/INSIDE
AMBULANCE CASE: HARASSMENT/INSIDE
INVESTIGATE/POSSIBLE CRIME: SUSP PERSON/OUTSIDE
AMBULANCE CASE: SERIOUS/INSIDE
LARCENY (PAST): PERSON/OUTSIDE (PROWLER)
TYP_DESC


TYP_DESC Counts for Month 3
Count
0
10000
20000
30000
40000
50000
60000
70000
80000
VISIBILITY PATROL: DIRECTED
STATION INSPECTION BY TRANSIT BUREAU PERSONNEL
SEE COMPLAINANT: OTHER/INSIDE
TRANSIT PATROL/INSPECTION BY NON-TRANSIT BUREAU PERSONNEL
TRAIN RUN/MOBILE ORDER MAINTENANCE SWEEP
AMBULANCE CASE: EDP/INSIDE
INVESTIGATE/POSSIBLE CRIME: CALLS FOR HELP/INSIDE
VISIBILITY PATROL: FAMILY/HOME VISIT
INVESTIGATE/POSSIBLE CRIME: SERIOUS/OTHER
INVESTIGATE/POSSIBLE CRIME: SUSP VEHICLE/OUTSIDE
VISIBILITY PATROL:PUBLIC/PRIVATE EDUCATIONAL FACILITY
ALARMS: COMMERCIAL/BURGLARY
VEHICLE ACCIDENT: SPECIAL CONDITION
DISPUTE: INSIDE
OTHER CRIMES (IN PROGRESS): FAMILY
TRAIN ORDER MAINTENANCE SWEEP (TOMS)
VISIBILITY PATROL: INTERIOR
DISORDERLY: PERSON/INSIDE
INVESTIGATE/POSSIBLE CRIME: SUSP PERSON/OUTSIDE (PROWLER)
AMBULANCE CASE: SERIOUS/INSIDE
AMBULANCE CASE: UNCONSCIOUS/INSIDE
INVESTIGATE/POSSIBLE CRIME: CALLS FOR HELP/OUTSIDE
LARCENY (PAST): OTHER/INSIDE
SEE COMPLAINANT: OTHER/OUTSIDE
AMBULANCE CASE: EDP/OUTSIDE
TYP_DESC


TYP_DESC Counts for Month 4
Count
0
10000
20000
30000
40000
50000
60000
70000
VISIBILITY PATROL: DIRECTED
STATION INSPECTION BY TRANSIT BUREAU PERSONNEL
SEE COMPLAINANT: OTHER/INSIDE
TRAIN RUN/MOBILE ORDER MAINTENANCE SWEEP
TRANSIT PATROL/INSPECTION BY NON-TRANSIT BUREAU PERSONNEL
INVESTIGATE/POSSIBLE CRIME: CALLS FOR HELP/INSIDE
AMBULANCE CASE: EDP/INSIDE
INVESTIGATE/POSSIBLE CRIME: SERIOUS/OTHER
VISIBILITY PATROL: FAMILY/HOME VISIT
INVESTIGATE/POSSIBLE CRIME: SUSP VEHICLE/OUTSIDE
ALARMS: COMMERCIAL/BURGLARY
VEHICLE ACCIDENT: SPECIAL CONDITION
TRAIN ORDER MAINTENANCE SWEEP (TOMS)
INVESTIGATE/POSSIBLE CRIME: SUSP PERSON/OUTSIDE (PROWLER)
OTHER CRIMES (IN PROGRESS): HARASSMENT/INSIDE
VISIBILITY PATROL: INTERIOR
VISIBILITY PATROL:PUBLIC/PRIVATE EDUCATIONAL FACILITY
DISORDERLY: PERSON/INSIDE
HOUSE OF WORSHIP VISIT
AMBULANCE CASE: EDP/OUTSIDE
INVESTIGATE/POSSIBLE CRIME: CALLS FOR HELP/OUTSIDE
AMBULANCE CASE: SERIOUS/INSIDE
AMBULANCE CASE: UNCONSCIOUS/INSIDE
LARCENY (PAST): OTHER/INSIDE
TYP_DESC


TYP_DESC Counts for Month 5
Count
0
10000
20000
30000
40000
50000
60000
70000
VISIBILITY PATROL: DIRECTED
STATION INSPECTION BY TRANSIT BUREAU PERSONNEL
SEE COMPLAINANT: OTHER/INSIDE
TRAIN RUN/MOBILE ORDER MAINTENANCE SWEEP
INVESTIGATE/POSSIBLE CRIME: CALLS FOR HELP/INSIDE
AMBULANCE CASE: EDP/INSIDE
INVESTIGATE/POSSIBLE CRIME: SERIOUS/OTHER
VISIBILITY PATROL: FAMILY/HOME VISIT
TRANSIT PATROL/INSPECTION BY NON-TRANSIT BUREAU PERSONNEL
INVESTIGATE/POSSIBLE CRIME: SUSP VEHICLE/OUTSIDE
VEHICLE ACCIDENT: SPECIAL CONDITION
ALARMS: COMMERCIAL/BURGLARY
DISPUTE: INSIDE
DISPUTE: FAMILY
VISIBILITY PATROL:PUBLIC/PRIVATE EDUCATIONAL FACILITY
INVESTIGATE/POSSIBLE CRIME: SUSP PERSON/OUTSIDE (PROWLER)
AMBULANCE CASE: EDP/OUTSIDE
TRAFFIC SAFETY
OTHER CRIMES (IN PROGRESS): HARASSMENT/INSIDE
INVESTIGATE/POSSIBLE CRIME: CALLS FOR HELP/OUTSIDE
VISIBILITY PATROL: INTERIOR
SEE COMPLAINANT: OTHER/OUTSIDE
AMBULANCE CASE: SERIOUS/INSIDE
TRAIN ORDER MAINTENANCE SWEEP (TOMS)
COMMUNITY TIME
LARCENY (PAST): OTHER/INSIDE
OTHER CRIMES (IN PROGRESS): HARASSMENT/OUTSIDE
DISORDERLY: PERSON/INSIDE
AMBULANCE CASE: UNCONSCIOUS/INSIDE
AMBULANCE CASE: UNCONSCIOUS/OUTSIDE
LARCENY (IN PROGRESS): OTHER/INSIDE
TYP_DESC



Joins the original DataFrame 'logs_1' with the previously filtered DataFrame 'ordered_filter_logs_1' based on the columns 'Month' and 'TYP_DESC.' Calculates the average response time ('ADD_TS_DISP_TS') for each incident type ('TYP_DESC').

```
from pyspark.sql.functions import avg

# Join the original DataFrame with the filtered one to get the relevant 'TYP_DESC' and 'Month'
logs_1_filtered = logs_1.join(ordered_filter_logs_1, ["Month", "TYP_DESC"])
# Calculate the average of ADD_TS_DISP_TS for each TYP_DESC
avg_disp_ts = logs_1_filtered.groupBy("TYP_DESC").agg(avg("ADD_TS_DISP_TS").alias("average_ADD_TS_DISP_TS"))

# Show the results
avg_disp_ts.show(truncate=False)
```

```
+--------------------------------------------------+---------------------+
|TYP_DESC                                          |average_ADD_TS_DISP_TS|
+--------------------------------------------------+---------------------+
|BUS INVESTIGATION                                 |3.955460244143847    |
|SEE COMPLAINANT: OTHER/TRANSIT                    |34.12704280155642    |
|DISORDERLY: PERSON/OUTSIDE                        |811.4935111322776    |
|AMBULANCE CASE: SERIOUS/TRANSIT                   |110.84109136006614   |
|HOUSE OF WORSHIP VISIT                            |1.6718883205456097   |
|VERIFY AMB NEEDED: TRANSIT                        |243.6395993387144    |
|LARCENY (PAST): OTHER/OUTSIDE                     |1215.1319199057714   |
|ASSAULT (PAST): OTHER/INSIDE                      |1131.3556951184698   |
|DISPUTE: OUTSIDE                                  |982.3184868519163    |
|VERIFY AMB NEEDED                                 |1086.9106797748088   |
|INVESTIGATE/POSSIBLE CRIME: SUSP PERSON/TRANSIT   |6.2446153846153845   |
|DISORDERLY: GROUP/OUTSIDE                         |967.1716235129461    |
|OTHER CRIMES (IN PROGRESS): HARASSMENT/OUTSIDE    |481.0286376653484    |
|LARCENY (PAST): VEHICLE/OUTSIDE                   |1235.56149372513     |
|VEHICLE ACCIDENT: HIT BY AUTO                     |543.3003445635528    |
|SHOT SPOTTER                                      |81.21824907521578    |
|INVESTIGATE/POSSIBLE CRIME: SERIOUS OTHER/LTD ACC HWY|960.020606060606     |
|VEHICLE ACCIDENT: INJURY                          |1061.9219585036153   |
```

Groups the DataFrame 'logs_1' by 'Month' and 'TYP_DESC,' counts the occurrences, filters the results to include only those with counts between 1000 and 5000, orders the filtered DataFrame by 'Month' and count in descending order.

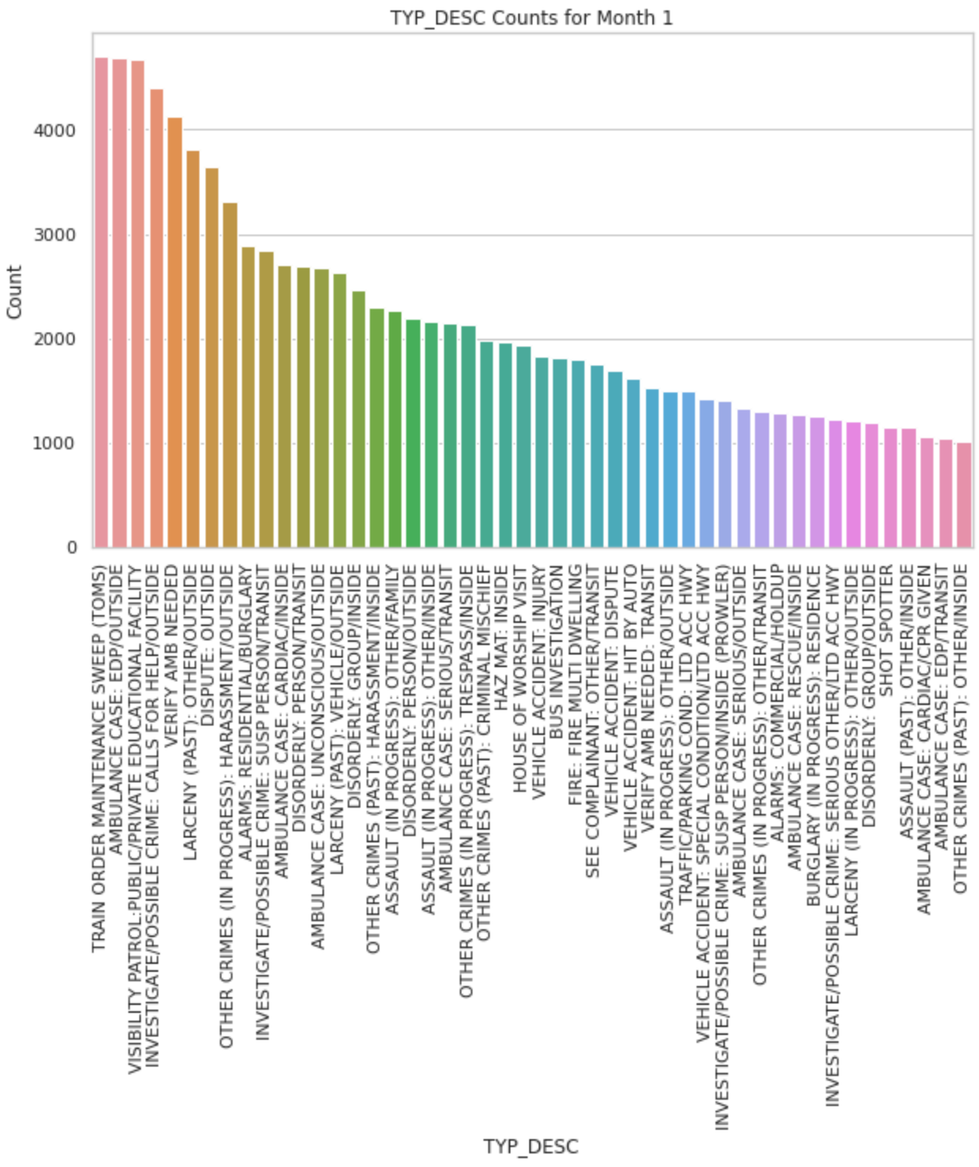
```
grouped_logs_1 = logs_1.groupBy("Month", "TYP_DESC").count()

# Filter to get only TYP_DESC with count between 5000 to 1000
filtered_logs_1 = grouped_logs_1.filter((grouped_logs_1['count'] >= 1000) & (grouped_logs_1['count'] <= 5000))

# Order by Month and count in descending order
ordered_filter_logs_1 = filtered_logs_1.orderBy("Month", F.col("count").desc())

# Show the results
ordered_filter_logs_1.show(truncate=False)
```

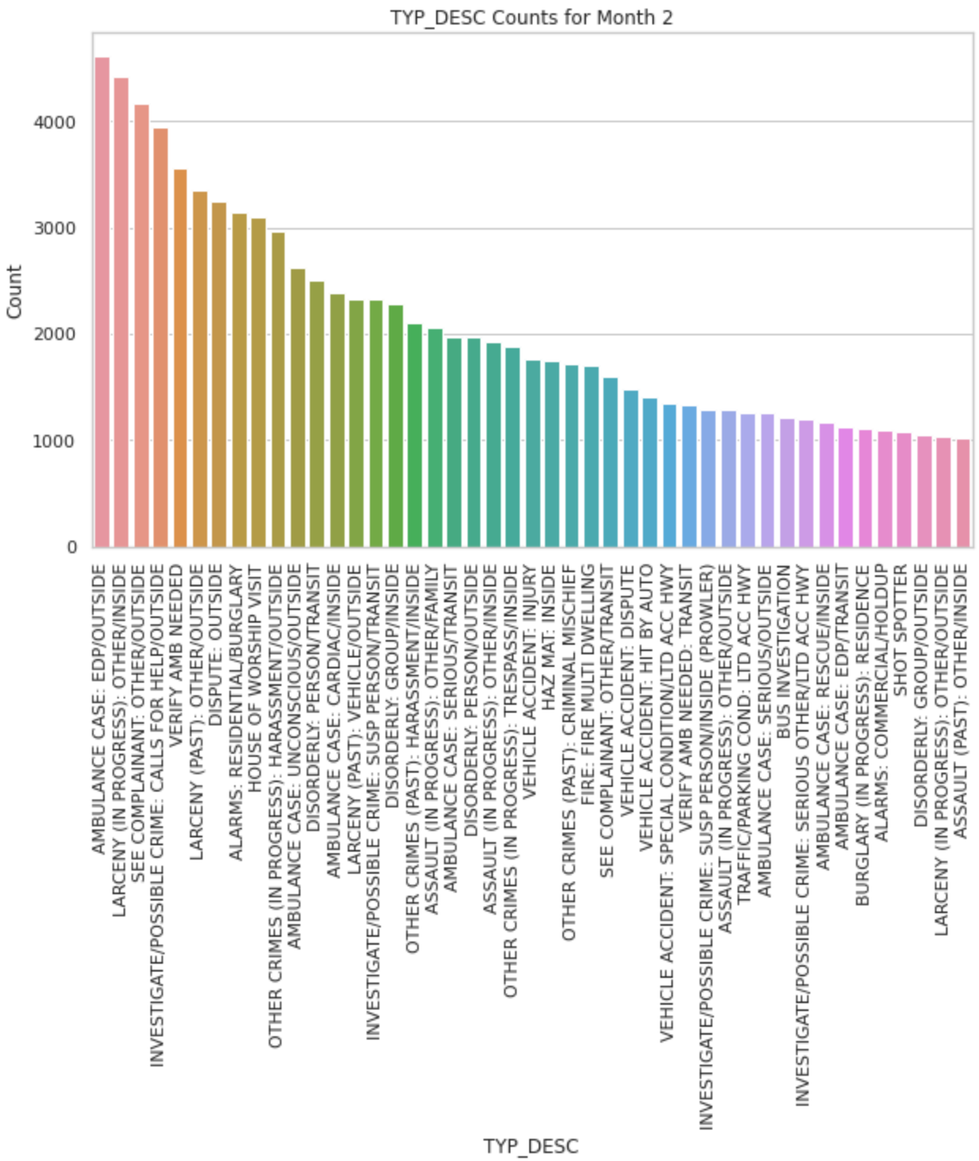
```
+-----+-----------------------------------------------------+-----+
|Month|TYP_DESC                                             |count|
+-----+-----------------------------------------------------+-----+
|1    |TRAIN ORDER MAINTENANCE SWEEP (TOMS)                 |4698 |
|1    |AMBULANCE CASE: EDP/OUTSIDE                          |4693 |
|1    |VISIBILITY PATROL:PUBLIC/PRIVATE EDUCATIONAL FACILITY|4676 |
|1    |INVESTIGATE/POSSIBLE CRIME: CALLS FOR HELP/OUTSIDE   |4397 |
|1    |VERIFY AMB NEEDED                                    |4129 |
|1    |LARCENY (PAST): OTHER/OUTSIDE                        |3809 |
|1    |DISPUTE: OUTSIDE                                     |3649 |
|1    |OTHER CRIMES (IN PROGRESS): HARASSMENT/OUTSIDE       |3318 |
|1    |ALARMS: RESIDENTIAL/BURGLARY                         |2894 |
|1    |INVESTIGATE/POSSIBLE CRIME: SUSP PERSON/TRANSIT      |2847 |
|1    |AMBULANCE CASE: CARDIAC/INSIDE                       |2712 |
|1    |DISORDERLY: PERSON/TRANSIT                           |2684 |
|1    |AMBULANCE CASE: UNCONSCIOUS/OUTSIDE                  |2681 |
|1    |LARCENY (PAST): VEHICLE/OUTSIDE                      |2637 |
|1    |DISORDERLY: GROUP/INSIDE                             |2458 |
|1    |OTHER CRIMES (PAST): HARASSMENT/INSIDE               |2295 |
|1    |ASSAULT (IN PROGRESS): OTHER/FAMILY                  |2273 |
|1    |DISORDERLY: PERSON/OUTSIDE                           |2194 |
```

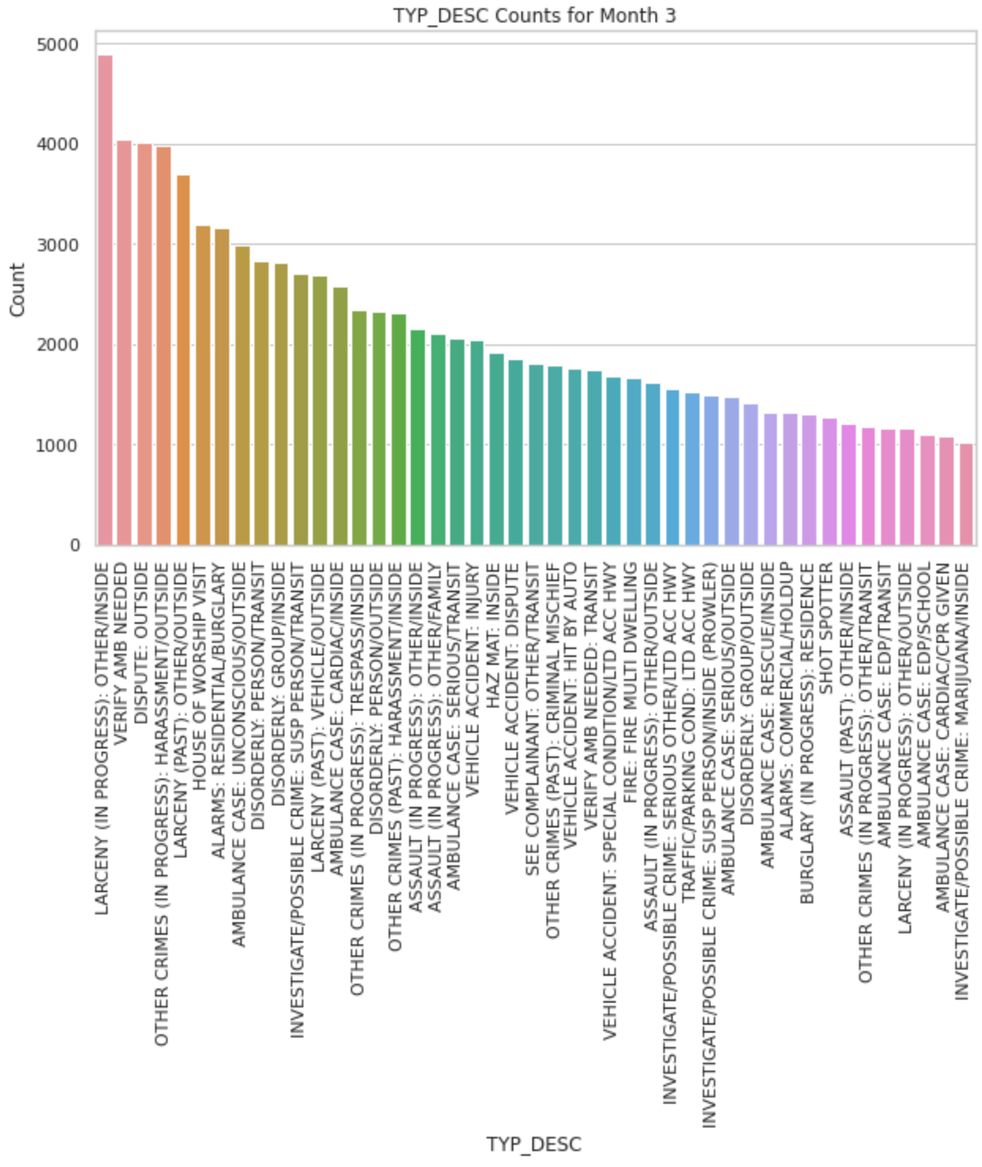
Filters the data, creates a bar plot with 'TYP_DESC' on the x-axis and count on the y-axis, and displays the plot with appropriate labels and title.

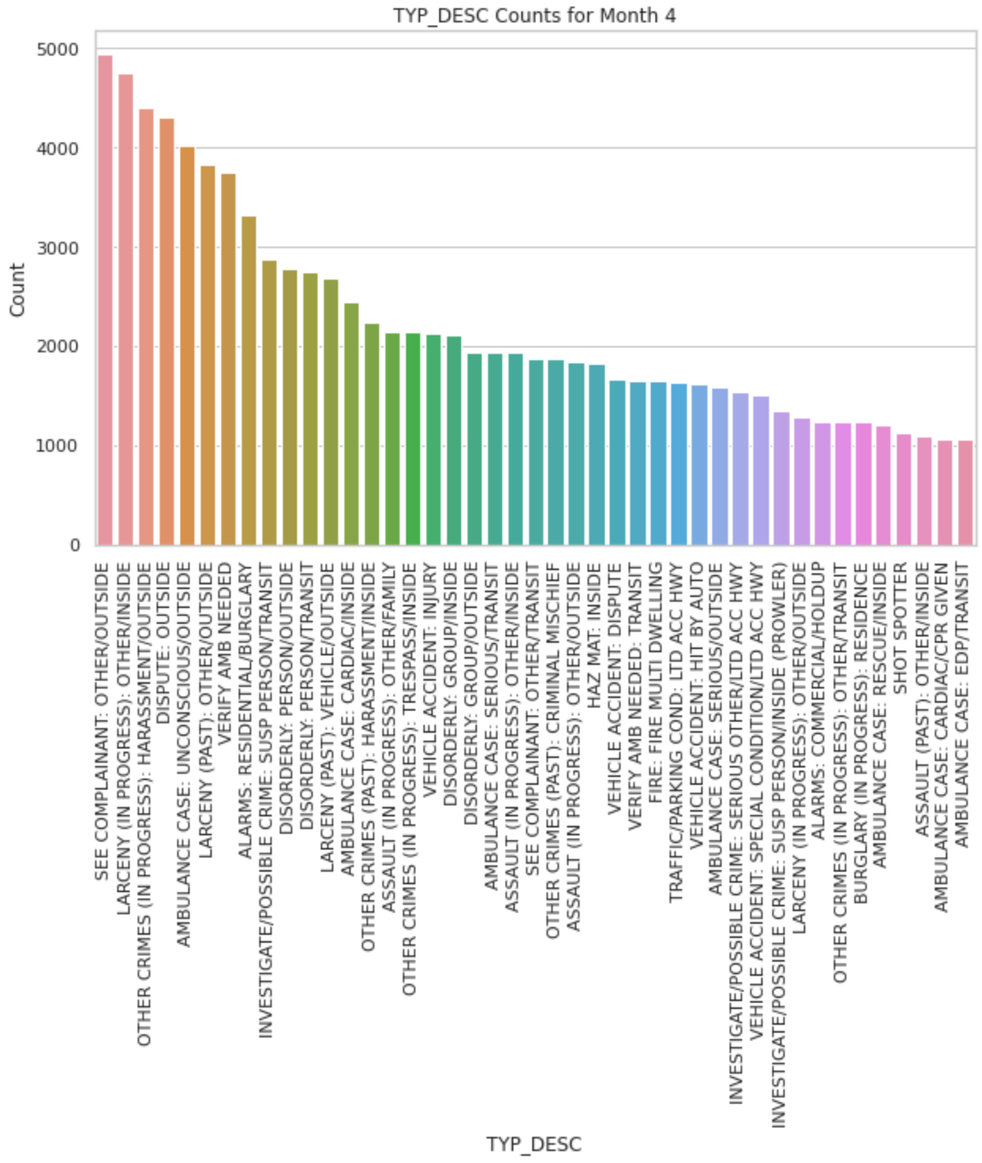
```
!pip install matplotlib seaborn
import matplotlib.pyplot as plt
import seaborn as sns
pandas_df = ordered_filter_logs_1.toPandas()
sns.set(style="whitegrid")

# Get the unique months from the DataFrame
unique_months = pandas_df['Month'].unique()

# Create a plot for each month
for month in unique_months:
    # Filter data for the specific month
    month_data = pandas_df[pandas_df['Month'] == month]

    # Create a bar plot
    plt.figure(figsize=(10, 6))
    sns.barplot(x='TYP_DESC', y='count', data=month_data)

    # Set plot title and labels
    plt.title(f'TYP_DESC Counts for Month {month}')
    plt.xlabel('TYP_DESC')
    plt.ylabel('Count')
    plt.xticks(rotation=90)  # Rotate the x labels for better readability

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

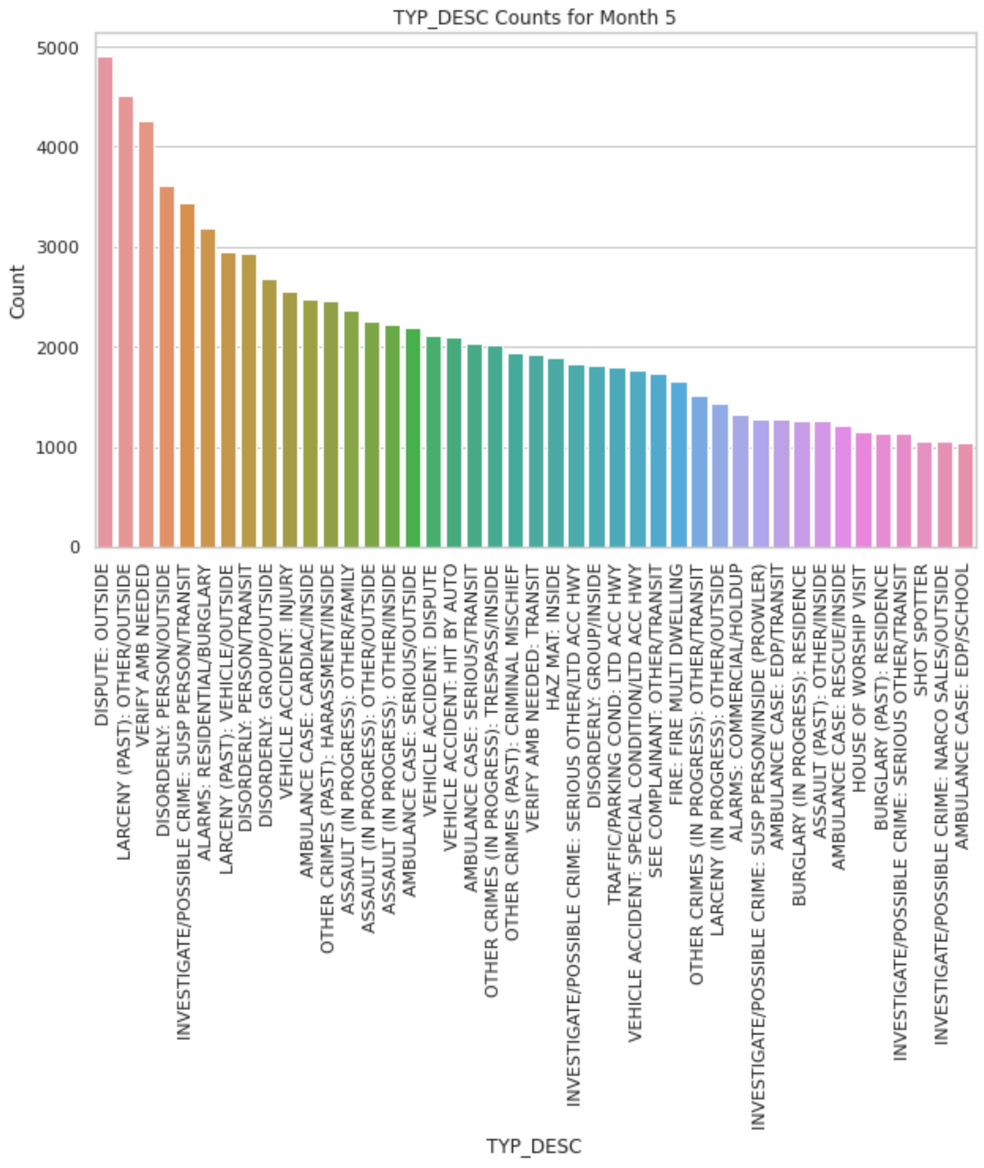
```
Requirement already satisfied: matplotlib in /databricks/python3/lib/python3.9/site-packages (3.5.1)
Requirement already satisfied: seaborn in /databricks/python3/lib/python3.9/site-packages (0.11.2)
Requirement already satisfied: packaging>=20.0 in /databricks/python3/lib/python3.9/site-packages (from matplotlib) (21.3)
Requirement already satisfied: python-dateutil>=2.7 in /databricks/python3/lib/python3.9/site-packages (from matplotlib) (2.8.2)
Requirement already satisfied: fonttools>=4.22.0 in /databricks/python3/lib/python3.9/site-packages (from matplotlib) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /databricks/python3/lib/python3.9/site-packages (from matplotlib) (1.3.2)
Requirement already satisfied: numpy>=1.17 in /databricks/python3/lib/python3.9/site-packages (from matplotlib) (1.21.5)
Requirement already satisfied: pyparsing>=2.2.1 in /databricks/python3/lib/python3.9/site-packages (from matplotlib) (3.0.4)
Requirement already satisfied: cycler>=0.10 in /databricks/python3/lib/python3.9/site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: pillow>=6.2.0 in /databricks/python3/lib/python3.9/site-packages (from matplotlib) (9.0.1)
Requirement already satisfied: scipy>=1.0 in /databricks/python3/lib/python3.9/site-packages (from seaborn) (1.7.3)
Requirement already satisfied: pandas>=0.23 in /databricks/python3/lib/python3.9/site-packages (from seaborn) (1.4.2)
Requirement already satisfied: pytz>=2020.1 in /databricks/python3/lib/python3.9/site-packages (from pandas>=0.23->seaborn) (2021.3)
Requirement already satisfied: six>=1.5 in /databricks/python3/lib/python3.9/site-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
WARNING: You are using pip version 21.2.4; however, version 23.3.2 is available.
You should consider upgrading via the '/local_disk0/.ephemeral_nfs/envs/pythonEnv-d5bfbe7c-8eb6-4808-a53e-4cc9dd6c8df1/bin/python -m pip install --upgrade pip' command.
```

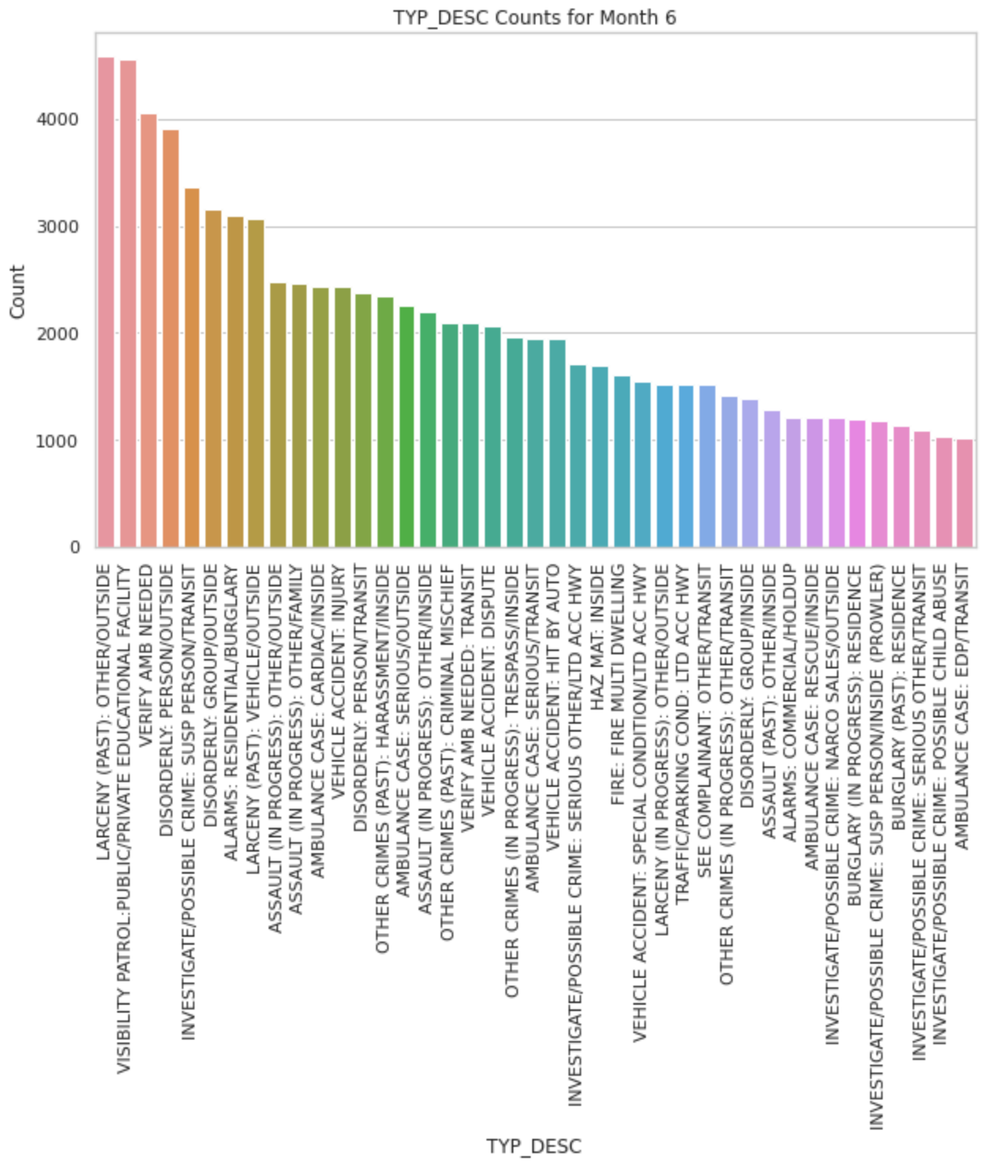

TYP_DESC Counts for Month 1
Count
0
1000
2000
3000
4000
TYP_DESC
TRAIN ORDER MAINTENANCE SWEEP (TOMS)
AMBULANCE CASE: EDP/OUTSIDE
VISIBILITY PATROL:PUBLIC/PRIVATE EDUCATIONAL FACILITY
INVESTIGATE/POSSIBLE CRIME: CALLS FOR HELP/OUTSIDE
VERIFY AMB NEEDED
LARCENY (PAST): OTHER/OUTSIDE
DISPUTE: OUTSIDE
OTHER CRIMES (IN PROGRESS): HARASSMENT/OUTSIDE
ALARMS: RESIDENTIAL/BURGLARY
INVESTIGATE/POSSIBLE CRIME: SUSP PERSON/TRANSIT
AMBULANCE CASE: CARDIAC/INSIDE
DISORDERLY: PERSON/TRANSIT
AMBULANCE CASE: UNCONSCIOUS/OUTSIDE
LARCENY (PAST): VEHICLE/OUTSIDE
DISORDERLY: GROUP/INSIDE
OTHER CRIMES (PAST): HARASSMENT/INSIDE
ASSAULT (IN PROGRESS): OTHER/FAMILY
DISORDERLY: PERSON/OUTSIDE
ASSAULT (IN PROGRESS): OTHER/INSIDE
AMBULANCE CASE: SERIOUS/TRANSIT
OTHER CRIMES (IN PROGRESS): TRESPASS/INSIDE
OTHER CRIMES (PAST): CRIMINAL MISCHIEF
HAZ MAT: INSIDE
HOUSE OF WORSHIP VISIT
VEHICLE ACCIDENT: INJURY
BUS INVESTIGATION
FIRE: FIRE MULTI DWELLING
SEE COMPLAINANT: OTHER/TRANSIT
VEHICLE ACCIDENT: DISPUTE
VEHICLE ACCIDENT: HIT BY AUTO
VERIFY AMB NEEDED: TRANSIT
ASSAULT (IN PROGRESS): OTHER/OUTSIDE
TRAFFIC/PARKING COND: LTD ACC HWY
VEHICLE ACCIDENT: SPECIAL CONDITION/LTD ACC HWY
INVESTIGATE/POSSIBLE CRIME: SUSP PERSON/INSIDE (PROWLER)
AMBULANCE CASE: SERIOUS/OUTSIDE
OTHER CRIMES (IN PROGRESS): OTHER/TRANSIT
ALARMS: COMMERCIAL/HOLDUP
AMBULANCE CASE: RESCUE/INSIDE
BURGLARY (IN PROGRESS): RESIDENCE/INSIDE
INVESTIGATE/POSSIBLE CRIME: SERIOUS OTHER/LTD ACC HWY
LARCENY (IN PROGRESS): OTHER/OUTSIDE
DISORDERLY: GROUP/OUTSIDE
SHOT SPOTTER
ASSAULT (PAST): OTHER/INSIDE
AMBULANCE CASE: CARDIAC/CPR GIVEN
AMBULANCE CASE: EDP/TRANSIT
OTHER CRIMES (PAST): OTHER/INSIDE


TYP_DESC Counts for Month 2
Count
0
1000
2000
3000
4000
AMBULANCE CASE: EDP/OUTSIDE
LARCENY (IN PROGRESS): OTHER/INSIDE
SEE COMPLAINANT: OTHER/OUTSIDE
INVESTIGATE/POSSIBLE CRIME: CALLS FOR HELP/OUTSIDE
VERIFY AMB NEEDED
LARCENY (PAST): OTHER/OUTSIDE
DISPUTE: OUTSIDE
ALARMS: RESIDENTIAL/BURGLARY
HOUSE OF WORSHIP VISIT
OTHER CRIMES (IN PROGRESS): HARASSMENT/OUTSIDE
AMBULANCE CASE: UNCONSCIOUS/OUTSIDE
DISORDERLY: PERSON/TRANSIT
AMBULANCE CASE: CARDIAC/INSIDE
LARCENY (PAST): VEHICLE/OUTSIDE
INVESTIGATE/POSSIBLE CRIME: SUSP PERSON/TRANSIT
DISORDERLY: GROUP/INSIDE
OTHER CRIMES (PAST): HARASSMENT/INSIDE
ASSAULT (IN PROGRESS): OTHER/FAMILY
AMBULANCE CASE: SERIOUS/TRANSIT
DISORDERLY: PERSON/OUTSIDE
ASSAULT (IN PROGRESS): OTHER/INSIDE
OTHER CRIMES (IN PROGRESS): TRESPASS/INSIDE
VEHICLE ACCIDENT: INJURY
HAZ MAT: INSIDE
OTHER CRIMES (PAST): CRIMINAL MISCHIEF
FIRE: FIRE MULTI DWELLING
SEE COMPLAINANT: OTHER/TRANSIT
VEHICLE ACCIDENT: DISPUTE
VEHICLE ACCIDENT: HIT BY AUTO
VEHICLE ACCIDENT: SPECIAL CONDITION/LTD ACC HWY
VERIFY AMB NEEDED: TRANSIT
INVESTIGATE/POSSIBLE CRIME: SUSP PERSON/INSIDE (PROWLER)
ASSAULT (IN PROGRESS): OTHER/OUTSIDE
TRAFFIC/PARKING COND: LTD ACC HWY
AMBULANCE CASE: SERIOUS/OUTSIDE
BUS INVESTIGATION
INVESTIGATE/POSSIBLE CRIME: SERIOUS OTHER/LTD ACC HWY
AMBULANCE CASE: RESCUE/INSIDE
AMBULANCE CASE: EDP/TRANSIT
BURGLARY (IN PROGRESS): RESIDENCE
ALARMS: COMMERCIAL/HOLDUP
SHOT SPOTTER
DISORDERLY: GROUP/OUTSIDE
LARCENY (IN PROGRESS): OTHER/OUTSIDE
ASSAULT (PAST): OTHER/INSIDE
TYP_DESC


TYP_DESC Counts for Month 3
Count
0
1000
2000
3000
4000
5000
TYP_DESC
LARCENY (IN PROGRESS): OTHER/INSIDE
VERIFY AMB NEEDED
DISPUTE: OUTSIDE
OTHER CRIMES (IN PROGRESS): HARASSMENT/OUTSIDE
LARCENY (PAST): OTHER/OUTSIDE
HOUSE OF WORSHIP VISIT
ALARMS: RESIDENTIAL/BURGLARY
AMBULANCE CASE: UNCONSCIOUS/OUTSIDE
DISORDERLY: PERSON/TRANSIT
DISORDERLY: GROUP/INSIDE
INVESTIGATE/POSSIBLE CRIME: SUSP PERSON/TRANSIT
LARCENY (PAST): VEHICLE/OUTSIDE
AMBULANCE CASE: CARDIAC/INSIDE
OTHER CRIMES (IN PROGRESS): TRESPASS/INSIDE
DISORDERLY: PERSON/OUTSIDE
OTHER CRIMES (PAST): HARASSMENT/INSIDE
ASSAULT (IN PROGRESS): OTHER/INSIDE
ASSAULT (IN PROGRESS): OTHER/FAMILY
AMBULANCE CASE: SERIOUS/TRANSIT
VEHICLE ACCIDENT: INJURY
HAZ MAT: INSIDE
VEHICLE ACCIDENT: DISPUTE
SEE COMPLAINANT: OTHER/TRANSIT
OTHER CRIMES (PAST): CRIMINAL MISCHIEF
VEHICLE ACCIDENT: HIT BY AUTO
VERIFY AMB NEEDED: TRANSIT
VEHICLE ACCIDENT: SPECIAL CONDITION/LTD ACC HWY
FIRE: FIRE MULTI DWELLING
ASSAULT (IN PROGRESS): OTHER/OUTSIDE
INVESTIGATE/POSSIBLE CRIME: SERIOUS OTHER/LTD ACC HWY
TRAFFIC/PARKING COND: LTD ACC HWY
INVESTIGATE/POSSIBLE CRIME: SUSP PERSON/INSIDE (PROWLER)
AMBULANCE CASE: SERIOUS/OUTSIDE
DISORDERLY: GROUP/OUTSIDE
AMBULANCE CASE: RESCUE/INSIDE
ALARMS: COMMERCIAL/HOLDUP
BURGLARY (IN PROGRESS): RESIDENCE
SHOT SPOTTER
ASSAULT (PAST): OTHER/INSIDE
OTHER CRIMES (IN PROGRESS): OTHER/TRANSIT
AMBULANCE CASE: EDP/TRANSIT
LARCENY (IN PROGRESS): OTHER/OUTSIDE
AMBULANCE CASE: EDP/SCHOOL
AMBULANCE CASE: CARDIAC/CPR GIVEN
INVESTIGATE/POSSIBLE CRIME: MARIJUANA/INSIDE


TYP_DESC Counts for Month 4
Count
0
1000
2000
3000
4000
5000
TYP_DESC
SEE COMPLAINANT: OTHER/OUTSIDE
LARCENY (IN PROGRESS): OTHER/INSIDE
OTHER CRIMES (IN PROGRESS): HARASSMENT/OUTSIDE
DISPUTE: OUTSIDE
AMBULANCE CASE: UNCONSCIOUS/OUTSIDE
LARCENY (PAST): OTHER/OUTSIDE
VERIFY AMB NEEDED
ALARMS: RESIDENTIAL/BURGLARY
INVESTIGATE/POSSIBLE CRIME: SUSP PERSON/TRANSIT
DISORDERLY: PERSON/OUTSIDE
DISORDERLY: PERSON/TRANSIT
LARCENY (PAST): VEHICLE/OUTSIDE
AMBULANCE CASE: CARDIAC/INSIDE
OTHER CRIMES (PAST): HARASSMENT/INSIDE
ASSAULT (IN PROGRESS): OTHER/FAMILY
OTHER CRIMES (IN PROGRESS): TRESPASS/INSIDE
VEHICLE ACCIDENT: INJURY
DISORDERLY: GROUP/INSIDE
DISORDERLY: GROUP/OUTSIDE
AMBULANCE CASE: SERIOUS/TRANSIT
ASSAULT (IN PROGRESS): OTHER/INSIDE
SEE COMPLAINANT: OTHER/TRANSIT
OTHER CRIMES (PAST): CRIMINAL MISCHIEF
ASSAULT (IN PROGRESS): OTHER/OUTSIDE
HAZ MAT: INSIDE
VEHICLE ACCIDENT: DISPUTE
VERIFY AMB NEEDED: TRANSIT
FIRE: FIRE MULTI DWELLING
TRAFFIC/PARKING COND: LTD ACC HWY
VEHICLE ACCIDENT: HIT BY AUTO
AMBULANCE CASE: SERIOUS/OUTSIDE
INVESTIGATE/POSSIBLE CRIME: SERIOUS OTHER/LTD ACC HWY
VEHICLE ACCIDENT: SPECIAL CONDITION/LTD ACC HWY
INVESTIGATE/POSSIBLE CRIME: SUSP PERSON/INSIDE (PROWLER)
LARCENY (IN PROGRESS): OTHER/OUTSIDE
ALARMS: COMMERCIAL/HOLDUP
OTHER CRIMES (IN PROGRESS): OTHER/TRANSIT
BURGLARY (IN PROGRESS): RESIDENCE
AMBULANCE CASE: RESCUE/INSIDE
SHOT SPOTTER
ASSAULT (PAST): OTHER/INSIDE
AMBULANCE CASE: CARDIAC/CPR GIVEN
AMBULANCE CASE: EDP/TRANSIT


TYP_DESC Counts for Month 5
Count
0
1000
2000
3000
4000
5000
TYP_DESC
DISPUTE: OUTSIDE
LARCENY (PAST): OTHER/OUTSIDE
VERIFY AMB NEEDED
DISORDERLY: PERSON/OUTSIDE
INVESTIGATE/POSSIBLE CRIME: SUSP PERSON/TRANSIT
ALARMS: RESIDENTIAL/BURGLARY
LARCENY (PAST): VEHICLE/OUTSIDE
DISORDERLY: PERSON/TRANSIT
DISORDERLY: GROUP/OUTSIDE
VEHICLE ACCIDENT: INJURY
AMBULANCE CASE: CARDIAC/INSIDE
OTHER CRIMES (PAST): HARASSMENT/INSIDE
ASSAULT (IN PROGRESS): OTHER/FAMILY
ASSAULT (IN PROGRESS): OTHER/OUTSIDE
ASSAULT (IN PROGRESS): OTHER/INSIDE
AMBULANCE CASE: SERIOUS/OUTSIDE
VEHICLE ACCIDENT: DISPUTE
AMBULANCE CASE: HIT BY AUTO
OTHER CRIMES (IN PROGRESS): SERIOUS/TRANSIT
OTHER CRIMES (PAST): TRESPASS/INSIDE
VERIFY AMB NEEDED: CRIMINAL MISCHIEF
HAZ MAT: TRANSIT
INVESTIGATE/POSSIBLE CRIME: SERIOUS OTHER/LTD ACC HWY
DISORDERLY: GROUP/INSIDE
TRAFFIC/PARKING COND: LTD ACC HWY
VEHICLE ACCIDENT: SPECIAL CONDITION/LTD ACC HWY
SEE COMPLAINANT: OTHER/TRANSIT
FIRE: FIRE MULTI DWELLING
OTHER CRIMES (IN PROGRESS): OTHER/TRANSIT
LARCENY (IN PROGRESS): OTHER/OUTSIDE
ALARMS: COMMERCIAL/HOLDUP
INVESTIGATE/POSSIBLE CRIME: SUSP PERSON/INSIDE (PROWLER)
AMBULANCE CASE: EDP/TRANSIT
BURGLARY (IN PROGRESS): RESIDENCE
ASSAULT (PAST): OTHER/INSIDE
AMBULANCE CASE: RESCUE/INSIDE
HOUSE OF WORSHIP VISIT
BURGLARY (PAST): RESIDENCE
INVESTIGATE/POSSIBLE CRIME: SERIOUS OTHER/TRANSIT
SHOT SPOTTER
INVESTIGATE/POSSIBLE CRIME: NARCO SALES/OUTSIDE
AMBULANCE CASE: EDP/SCHOOL



Performs a join operation between the original DataFrame 'logs_1' and the filtered DataFrame 'ordered_filter_logs_1' based on 'Month' and 'TYP_DESC' columns, followed by calculating the average of the 'ADD_TS_DISP_TS' column for each 'TYP_DESC' and displaying the results.

```
from pyspark.sql.functions import avg

# Join the original DataFrame with the filtered one to get the relevant 'TYP_DESC' and 'Month'
logs_1_filtered = logs_1.join(ordered_filter_logs_1, ["Month", "TYP_DESC"])
# Calculate the average of ADD_TS_DISP_TS for each TYP_DESC
avg_disp_ts = logs_1_filtered.groupBy("TYP_DESC").agg(avg("ADD_TS_DISP_TS").alias("average_ADD_TS_DISP_TS"))

# Show the results
avg_disp_ts.show(truncate=False)
```

```
+-----------------------------------------------------+----------------------+
|TYP_DESC                                             |average_ADD_TS_DISP_TS|
+-----------------------------------------------------+----------------------+
|BUS INVESTIGATION                                    |3.955460244143847     |
|SEE COMPLAINANT: OTHER/TRANSIT                       |34.12704280155642     |
|DISORDERLY: PERSON/OUTSIDE                           |811.4935111322776     |
|AMBULANCE CASE: SERIOUS/TRANSIT                      |110.84109136006614    |
|HOUSE OF WORSHIP VISIT                               |1.6718883205456097    |
|VERIFY AMB NEEDED: TRANSIT                           |243.6395993387144     |
|LARCENY (PAST): OTHER/OUTSIDE                        |1215.1319199057714    |
|ASSAULT (PAST): OTHER/INSIDE                         |1131.3556951184698    |
|DISPUTE: OUTSIDE                                     |982.3184868519163     |
|VERIFY AMB NEEDED                                    |1086.9106797748088    |
|INVESTIGATE/POSSIBLE CRIME: SUSP PERSON/TRANSIT      |6.2446153846153845    |
|DISORDERLY: GROUP/OUTSIDE                            |967.1716235129461     |
|OTHER CRIMES (IN PROGRESS): HARASSMENT/OUTSIDE       |481.0286376653484     |
|LARCENY (PAST): VEHICLE/OUTSIDE                      |1235.56149372513      |
|VEHICLE ACCIDENT: HIT BY AUTO                        |543.3003445635528     |
|SHOT SPOTTER                                         |81.21824907521578     |
|INVESTIGATE/POSSIBLE CRIME: SERIOUS OTHER/LTD ACC HWY|960.020606060606      |
|VEHICLE ACCIDENT: INJURY                             |1061.9219585036153    |
```

Performs timestamp conversion on the DataFrame 'logs_1', assuming the date-time columns ('ADD_TS', 'DISP_TS', 'CLOSNG_TS') are in the 'MM/dd/yyyy hh:mm:ss a' format.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, unix_timestamp

# Initialize Spark Session
spark = SparkSession.builder.appName("TimestampConversion").getOrCreate()

# Assuming the format of your date-time columns is 'MM/dd/yyyy hh:mm:ss a'
# Convert to timestamp format
logs_1 = logs_1.withColumn("ADD_TS", unix_timestamp(col("ADD_TS"), 'MM/dd/yyyy hh:mm:ss a'))
logs_1 = logs_1.withColumn("DISP_TS", unix_timestamp(col("DISP_TS"), 'MM/dd/yyyy hh:mm:ss a'))
logs_1 = logs_1.withColumn("CLOSNG_TS", unix_timestamp(col("CLOSNG_TS"), 'MM/dd/yyyy hh:mm:ss a'))

# Show the resulting DataFrame
logs_1.show()
```

```
+-----------+-----------+-------------+-------------+-----------+---------+--------------------+--------+--------+--------------------+------------+----------+----------+----------+-
----+
|CAD_EVNT_ID|CREATE_DATE|INCIDENT_DATE|INCIDENT_TIME|NYPD_PCT_CD|  BORO_NM|       PATRL_BORO_NM|GEO_CD_X|GEO_CD_Y|           TYP_DESC|    CIP_JOBS|    ADD_TS|   DISP_TS| CLOSNG_TS|M
onth|
+-----------+-----------+-------------+-------------+-----------+---------+--------------------+--------+--------+--------------------+------------+----------+----------+----------+-
----+
|   91250176| 2023-01-01|   2022-12-31|     23:24:39|         67| BROOKLYN|PATROL BORO BKLYN...| 1001878|  175994|VEHICLE ACCIDENT:...|     Non CIP|1672535301|1672535397|1672538264|
1|
|   91250180| 2023-01-01|   2022-12-31|     23:24:47|         75| BROOKLYN|PATROL BORO BKLYN...| 1017204|  180778|ALARMS: COMMERCIA...|     Non CIP|1672533480|1672533514|1672537521|
1|
|   91250681| 2023-01-01|   2022-12-31|     23:55:56|        114|   QUEENS|PATROL BORO QUEEN...| 1008573|  217117|ALARMS: RESIDENTI...|     Non CIP|1672531286|1672531578|1672531587|
1|
|   91250683| 2023-01-01|   2022-12-31|     23:55:59|         66| BROOKLYN|PATROL BORO BKLYN...|  993234|  161780|ALARMS: RESIDENTI...|     Non CIP|1672531294|1672533434|1672536074|
1|
|   91250700| 2023-01-01|   2022-12-31|     23:57:08|        115|   QUEENS|PATROL BORO QUEEN...| 1014264|  211852|ALARMS: COMMERCIA...|     Non CIP|1672531289|1672532068|1672536262|
1|
|   91250736| 2023-01-01|   2022-12-31|     23:59:09|         46|    BRONX|   PATROL BORO BRONX| 1007356|  248923|ALARMS: COMMERCIA...|     Non CIP|1672531295|1672540824|1672583179|
1|
|   91250746| 2023-01-01|   2023-01-01|     00:00:12|          5|MANHATTAN|PATROL BORO MAN S...|  983903|  200257|SEE COMPLAINANT: ...|     Non CIP|1672531212|1672531216|1672538813|
1|
|   91250747| 2023-01-01|   2023-01-01|     00:00:15|         14|MANHATTAN|PATROL BORO MAN S...|  987608|  215185|INVESTIGATE/POSSI...|     Non CIP|1672531263|1672542390|1672542402|
```

Calculates the time differences in seconds between various timestamp columns ('ADD_TS', 'DISP_TS', 'CLOSNG_TS') and creates new columns ('ADD_TS_DISP_TS', 'DISP_TS_CLOSNG_TS', 'ADD_TS_CLOSNG_TS') to store these differences.

```
# Calculate the differences in seconds
logs_1 = logs_1.withColumn("ADD_TS_DISP_TS", col("DISP_TS") - col("ADD_TS"))
logs_1 = logs_1.withColumn("DISP_TS_CLOSNG_TS", col("CLOSNG_TS") - col("DISP_TS"))
logs_1 = logs_1.withColumn("ADD_TS_CLOSNG_TS", col("CLOSNG_TS") - col("ADD_TS"))
logs_1.show()
```

```
+-----------+-----------+-------------+-------------+-----------+---------+--------------------+--------+--------+--------------------+------------+----------+----------+----------+-
----+--------------+-----------------+-----------------+
|CAD_EVNT_ID|CREATE_DATE|INCIDENT_DATE|INCIDENT_TIME|NYPD_PCT_CD|  BORO_NM|       PATRL_BORO_NM|GEO_CD_X|GEO_CD_Y|           TYP_DESC|    CIP_JOBS|    ADD_TS|   DISP_TS| CLOSNG_TS|M
onth|ADD_TS_DISP_TS|DISP_TS_CLOSNG_TS|ADD_TS_CLOSNG_TS|
+-----------+-----------+-------------+-------------+-----------+---------+--------------------+--------+--------+--------------------+------------+----------+----------+----------+-
----+--------------+-----------------+-----------------+
|   91250176| 2023-01-01|   2022-12-31|     23:24:39|         67| BROOKLYN|PATROL BORO BKLYN...| 1001878|  175994|VEHICLE ACCIDENT:...|     Non CIP|1672535301|1672535397|1672538264|
1|            96|             2867|             2963|
|   91250180| 2023-01-01|   2022-12-31|     23:24:47|         75| BROOKLYN|PATROL BORO BKLYN...| 1017204|  180778|ALARMS: COMMERCIA...|     Non CIP|1672533480|1672533514|1672537521|
1|            34|             4007|             4041|
|   91250681| 2023-01-01|   2022-12-31|     23:55:56|        114|   QUEENS|PATROL BORO QUEEN...| 1008573|  217117|ALARMS: RESIDENTI...|     Non CIP|1672531286|1672531578|1672531587|
1|           292|                9|              301|
|   91250683| 2023-01-01|   2022-12-31|     23:55:59|         66| BROOKLYN|PATROL BORO BKLYN...|  993234|  161780|ALARMS: RESIDENTI...|     Non CIP|1672531294|1672533434|1672536074|
1|          2140|             2640|             4780|
|   91250700| 2023-01-01|   2022-12-31|     23:57:08|        115|   QUEENS|PATROL BORO QUEEN...| 1014264|  211852|ALARMS: COMMERCIA...|     Non CIP|1672531289|1672532068|1672536262|
1|           779|             4194|             4973|
|   91250736| 2023-01-01|   2022-12-31|     23:59:09|         46|    BRONX|   PATROL BORO BRONX| 1007356|  248923|ALARMS: COMMERCIA...|     Non CIP|1672531295|1672540824|1672583179|
1|          9529|            42355|            51884|
|   91250746| 2023-01-01|   2023-01-01|     00:00:12|          5|MANHATTAN|PATROL BORO MAN S...|  983903|  200257|SEE COMPLAINANT: ...|     Non CIP|1672531212|1672531216|1672538813|
1|             4|             7597|             7601|
|   91250747| 2023-01-01|   2023-01-01|     00:00:15|         14|MANHATTAN|PATROL BORO MAN S...|  987608|  215185|INVESTIGATE/POSSI...|     Non CIP|1672531263|1672542390|1672542402|
```

Defines a pipeline for building a linear regression model. It uses StringIndexer to convert categorical columns ('BORO_NM' and 'TYP_DESC') into numerical indices and employs VectorAssembler to combine these indices into a single feature vector column. pipeline is then fitted to the input DataFrame (logs_1), transforming the data, and creating a new DataFrame (df_transformed).

```
from pyspark.sql import SparkSession
from pyspark.ml.feature import StringIndexer, VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import RegressionEvaluator

# Initialize Spark Session
spark = SparkSession.builder.appName("LinearRegressionModel").getOrCreate()


# StringIndexer for categorical columns
indexer1 = StringIndexer(inputCol="BORO_NM", outputCol="BORO_NM_Index")
indexer2 = StringIndexer(inputCol="TYP_DESC", outputCol="TYP_DESC_Index")

# VectorAssembler to combine feature columns into a single vector column
assembler = VectorAssembler(inputCols=["BORO_NM_Index", "TYP_DESC_Index"], outputCol="features")

# Pipeline
pipeline = Pipeline(stages=[indexer1, indexer2, assembler])
pipelineModel = pipeline.fit(logs_1)
df_transformed = pipelineModel.transform(logs_1)
```

Splits the transformed data (df_transformed) into training and test sets, initializes a linear regression model, and trains the model using the training data. Makes predictions on the test data and evaluates the model's performance using metrics such as Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared (R^2).

```
# Split the data into training and test sets
train_data, test_data = df_transformed.randomSplit([0.7, 0.3])

# Initialize the Linear Regression model
lr = LinearRegression(featuresCol='features', labelCol='ADD_TS_DISP_TS')

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

# Make predictions
predictions = lr_model.transform(test_data)
```

```
from pyspark.ml.evaluation import RegressionEvaluator

# Evaluate the model
evaluator_rmse = RegressionEvaluator(labelCol="ADD_TS_DISP_TS", predictionCol="prediction", metricName="rmse")
evaluator_mae = RegressionEvaluator(labelCol="ADD_TS_DISP_TS", predictionCol="prediction", metricName="mae")
evaluator_r2 = RegressionEvaluator(labelCol="ADD_TS_DISP_TS", predictionCol="prediction", metricName="r2")

rmse = evaluator_rmse.evaluate(predictions)
mae = evaluator_mae.evaluate(predictions)
r2 = evaluator_r2.evaluate(predictions)

print("Root Mean Squared Error (RMSE) on test data =", rmse)
print("Mean Absolute Error (MAE) on test data =", mae)
print("R-squared (R²) on test data =", r2)
```

```
Root Mean Squared Error (RMSE) on test data = 1319.231575526898
Mean Absolute Error (MAE) on test data = 533.4505288301402
R-squared (R²) on test data = 0.01215453818023271
```

This function, make_prediction, takes user inputs for "BORO_NM" and "TYP_DESC," applies the necessary StringIndexer transformations, assembles the features using a pre-defined VectorAssembler, and then uses a trained Linear Regression model (lr_model) to make a prediction.

```
def make_prediction(boro_nm_input, typ_desc_input, boro_indexer, typ_indexer, assembler, lr_model, spark):
    """
    Make a prediction based on user inputs.

    Parameters:
    boro_nm_input (str): Input for BORO_NM.
    typ_desc_input (str): Input for TYP_DESC.
    boro_indexer (StringIndexer): StringIndexer for BORO_NM.
    typ_indexer (StringIndexer): StringIndexer for TYP_DESC.
    assembler (VectorAssembler): VectorAssembler that combines the indexed features.
    lr_model (LinearRegressionModel): Trained Linear Regression model.
    spark (SparkSession): Spark session object.

    Returns:
    float: Predicted value.
    """
    # Create a DataFrame from the input
    new_data = spark.createDataFrame([(boro_nm_input, typ_desc_input)], ["BORO_NM", "TYP_DESC"])

    # Fit the StringIndexers on the entire dataset to create StringIndexerModels
    boro_indexer_model = indexer1.fit(new_data)
    typ_indexer_model = indexer2.fit(new_data)

    # Apply the StringIndexer transformations
    new_data_indexed = boro_indexer_model.transform(new_data)
    new_data_indexed = typ_indexer_model.transform(new_data_indexed)

    # Assemble the features
    new_data_assembled = assembler.transform(new_data_indexed)

    # Make the prediction
    prediction = lr_model.transform(new_data_assembled)

    # Return the predicted value
    return prediction.select("prediction").collect()[0]['prediction']
```

The function is demonstrated using user inputs for borough ("QUEENS") and incident type ("PATROL BORO MAN SOUTH"), showcasing how the function can be applied to obtain a prediction based on these inputs.

```
# Example user inputs
user_input_boro = "QUEENS"
user_input_typ = "PATROL BORO MAN SOUTH"

# Assuming the following objects are already defined:
# boro_indexer_model, typ_indexer_model, vector_assembler, lr_model

# Make a prediction
predicted_value = make_prediction(user_input_boro, user_input_typ, indexer1, indexer2, assembler, lr_model, spark)

print(f"The predicted value is: {predicted_value}")
```

```
The predicted value is: 196.3088900895523
```

Creation of StringIndexer models (boro_indexer_model and typ_indexer_model) to transform categorical columns ("BORO_NM" and "TYP_DESC") into numerical indices. It then showcases the usage of these models, along with a pre-defined VectorAssembler (vector_assembler), to prepare features for making predictions using the make_prediction function.

```
from pyspark.ml.feature import StringIndexer

# StringIndexer for BORO_NM column
indexer_boro = StringIndexer(inputCol="BORO_NM", outputCol="BORO_NM_Index")
boro_indexer_model = indexer_boro.fit(logs_1)

# StringIndexer for TYP_DESC column
indexer_typ = StringIndexer(inputCol="TYP_DESC", outputCol="TYP_DESC_Index")
typ_indexer_model = indexer_typ.fit(logs_1)

predicted_value = make_prediction(user_input_boro, user_input_typ, boro_indexer_model, typ_indexer_model, assembler,
lr_model, spark)
print(f"The predicted value is: {predicted_value}")

from pyspark.ml.feature import VectorAssembler

# Assuming 'df' is your DataFrame and you've already indexed your categorical columns

# List of your feature columns (including the indexed categorical columns and other numerical features)
feature_columns = ['BORO_NM_Index', 'TYP_DESC_Index', 'OtherFeature1', 'OtherFeature2']  # Update with your actual
columns

# Create the VectorAssembler instance
vector_assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")
```

```
The predicted value is: 196.3088900895523
```

Calculates the pairwise correlation between all numeric columns in the DataFrame logs_1 using the corr function. It iterates over combinations of numeric columns and prints the correlation values, generating a correlation matrix.

```
from pyspark.sql.functions import corr
spark = SparkSession.builder.appName("CorrelationMatrix").getOrCreate()
# Assuming df is your DataFrame and you want to calculate correlations for all numeric columns
numeric_columns = [col_name for col_name, dtype in logs_1.dtypes if dtype in ['int', 'double']]

# Initialize an empty dictionary to store correlation values
correlation_matrix = {}

# Calculate pairwise correlations
for col1 in numeric_columns:
    for col2 in numeric_columns:
        correlation_value = logs_1.select(corr(col1, col2)).collect()[0][0]
        correlation_matrix[(col1, col2)] = correlation_value
# Print the correlation matrix
for key, value in correlation_matrix.items():
    print(f"Correlation between {key[0]} and {key[1]}: {value}")
```

```
Correlation between CAD_EVNT_ID and CAD_EVNT_ID: 1.0
Correlation between CAD_EVNT_ID and NYPD_PCT_CD: 0.0054443589129880635
Correlation between CAD_EVNT_ID and GEO_CD_X: 0.006833648021243556
Correlation between CAD_EVNT_ID and GEO_CD_Y: 0.006715922832387029
Correlation between CAD_EVNT_ID and Month: 0.9858972993620089
Correlation between NYPD_PCT_CD and CAD_EVNT_ID: 0.0054443589129880635
Correlation between NYPD_PCT_CD and NYPD_PCT_CD: 1.0
Correlation between NYPD_PCT_CD and GEO_CD_X: 0.26992191308926505
Correlation between NYPD_PCT_CD and GEO_CD_Y: -0.4701928260731668
Correlation between NYPD_PCT_CD and Month: 0.005634435941486121
Correlation between GEO_CD_X and CAD_EVNT_ID: 0.006833648021243556
Correlation between GEO_CD_X and NYPD_PCT_CD: 0.26992191308926505
Correlation between GEO_CD_X and GEO_CD_X: 1.0
Correlation between GEO_CD_X and GEO_CD_Y: 0.2887251058845572
Correlation between GEO_CD_X and Month: 0.006192443672665521
Correlation between GEO_CD_Y and CAD_EVNT_ID: 0.00671592283238703
Correlation between GEO_CD_Y and NYPD_PCT_CD: -0.47019282607316676
Correlation between GEO_CD_Y and GEO_CD_X: 0.2887251058845572
Correlation between GEO_CD_Y and GEO_CD_Y: 1.0
Correlation between GEO_CD_Y and Month: 0.005920635271772752
Correlation between Month and CAD_EVNT_ID: 0.985897299362009
```

```
import pandas as pd

# Convert the correlation dictionary to a Pandas DataFrame
correlation_df = pd.DataFrame(correlation_matrix).apply(pd.Series)

# The DataFrame is currently in a 'long' format, convert it to a 'wide' format
correlation_df.index = pd.MultiIndex.from_tuples(correlation_df.index)
correlation_df = correlation_df.unstack().reset_index(level=0, drop=True)

# Ensure the DataFrame is in the correct format
correlation_df.columns = correlation_df.index

!pip install seaborn matplotlib
import seaborn as sns
import matplotlib.pyplot as plt

# Set up the matplotlib figure
plt.figure(figsize=(10, 8))

# Draw the heatmap
sns.heatmap(correlation_df.astype(float), annot=True, fmt=".2f", cmap='coolwarm')

# Add title and labels
plt.title('Correlation Matrix Heatmap')
plt.xlabel('Features')
plt.ylabel('Features')

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

```
ValueError: If using all scalar values, you must pass an index
```

Calculates the mean value for the column "ADD_TS_DISP_TS" in the DataFrame logs_1 using the mean function from PySpark's functions module. Computes the median (approximate) for the same column using the approxQuantile method, specifying a quantile array with the median position (0.5) and an acceptable relative error of 0.01.

```
from pyspark.sql.functions import mean, col
# Calculate mean for ADD_TS_DISP_TS
mean_value = logs_1.agg(mean(col("ADD_TS_DISP_TS"))).collect()[0][0]
print(f"Mean of ADD_TS_DISP_TS: {mean_value}")

# Calculate median (approximate) for ADD_TS_DISP_TS
median_value = logs_1.approxQuantile("ADD_TS_DISP_TS", [0.5], 0.01)[0]
print(f"Median of ADD_TS_DISP_TS: {median_value}")
```

```
Mean of ADD_TS_DISP_TS: 371.5924402183266
Median of ADD_TS_DISP_TS: 3.0
```

Calculates pairwise correlations between numeric columns in a PySpark DataFrame (logs_1). It iterates through all unique pairs of numeric columns, using the corr method from the stat.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import corr

# Initialize Spark Session
spark = SparkSession.builder.appName("CorrelationMatrix").getOrCreate()

# Assuming df is your DataFrame loaded with the above schema
numeric_columns = ['CAD_EVNT_ID', 'NYPD_PCT_CD', 'GEO_CD_X', 'GEO_CD_Y', 'ADD_TS',
                   'DISP_TS', 'CLOSNG_TS', 'Month', 'ADD_TS_DISP_TS',
                   'DISP_TS_CLOSNG_TS', 'ADD_TS_CLOSNG_TS']

# Calculate pairwise correlations
correlation_values = []
for i in range(len(numeric_columns)):
    for j in range(i, len(numeric_columns)):
        corr_val = logs_1.stat.corr(numeric_columns[i], numeric_columns[j])
        correlation_values.append((numeric_columns[i], numeric_columns[j], corr_val))
```

Converts the list of tuples, containing column pairs and their correlation values, into a Pandas DataFrame named correlation_df with three columns.

Converts the list of tuples, representing column pairs and their correlation values. Pandas pivot function is then applied to reshape this DataFrame into a square matrix, where row and column indices represent the columns involved, and the matrix elements contain the corresponding correlation coefficients.

```
 import pandas as pd

# Convert the list of tuples into a Pandas DataFrame
correlation_df = pd.DataFrame(correlation_values, columns=["Column1", "Column2", "Correlation"])

# Pivot the DataFrame to create a square matrix
correlation_matrix = correlation_df.pivot(index='Column1', columns='Column2', values='Correlation')
```

```
Generate a heatmap visualization of the correlation matrix (correlation_matrix).
```