Lab3_python

November 24, 2022

[1]:
```
import findspark
findspark.init()
```

[2]:
```
# configure spark variables
from pyspark.context import SparkContext
from pyspark.sql.context import SQLContext
from pyspark.sql.session import SparkSession

sc = SparkContext()
sqlContext = SQLContext(sc)
spark = SparkSession(sc)

# load up other dependencies
import re
import pandas as pd
```

[3]:
```
m = re.finditer(r'.*?(spark).*?', "I'm searching for a spark in PySpark", re.I)
for match in m:
    print(match, match.start(), match.end())
```

```
<re.Match object; span=(0, 25), match="I'm searching for a spark"> 0 25
<re.Match object; span=(25, 36), match=' in PySpark'> 25 36
```

[4]:
```
raw_data_files = ['/home/spark/files/access_log1', '/home/spark/files/
 ↪access_log2']
base_df = spark.read.text(raw_data_files)
base_df.printSchema()
```

```
root
 |-- value: string (nullable = true)
```

[5]:
```
type(base_df)
```

[5]:
```
pyspark.sql.dataframe.DataFrame
```

[6]:
```
base_df.show(5, truncate=False)
```

```
+------------------------------------------------------------------------------
------------------------------------------------------------------------------
-------------------------------------------------------------------+
|value
|
+------------------------------------------------------------------------------
------------------------------------------------------------------------------
-------------------------------------------------------------------+
|2001:6d8:10:4400:451:aebb:715b:b6df - - [10/Nov/2019:03:44:13 +0100] "GET
/wpad.dat HTTP/1.1" 304 - "-" "-"
|
|2001:6d8:10:4400::1:100b - - [10/Nov/2019:03:44:13 +0100] "GET /wpad.dat
HTTP/1.1" 200 137 "-" "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36
(KHTML, like Gecko) ReaderServices/19.12.20036 Chrome/80.0.0.0 Safari/537.36"|
|172.30.254.52 - - [10/Nov/2019:03:44:15 +0100] "GET /wpad.dat HTTP/1.1" 200 137
"-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
ReaderServices/19.12.20036 Chrome/80.0.0.0 Safari/537.36"            |
|2001:6d8:10:4400::1:100b - - [10/Nov/2019:03:44:21 +0100] "GET /wpad.dat
HTTP/1.1" 200 137 "-" "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36
(KHTML, like Gecko) ReaderServices/19.12.20036 Chrome/80.0.0.0 Safari/537.36"|
|172.30.254.52 - - [10/Nov/2019:03:44:23 +0100] "GET /wpad.dat HTTP/1.1" 200 137
"-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
ReaderServices/19.12.20036 Chrome/80.0.0.0 Safari/537.36"            |
+------------------------------------------------------------------------------
------------------------------------------------------------------------------
-------------------------------------------------------------------+
only showing top 5 rows
```

[7]:
```
base_df_rdd = base_df.rdd
```

[8]:
```
from pyspark.sql.functions import regexp_extract

log_df = base_df.select(regexp_extract('value', r'(^[\S+]+) -', 1).
 ↪alias('host'),
                        regexp_extract('value', r'(^[\S+]+) - - \[(.*)\]', 2).
 ↪alias('timestamp'),
                        regexp_extract('value', r'(^[\S+]+) - - \[(.*)\]␣
 ↪"(\w+)', 3).alias('method'),
                        regexp_extract('value', r'(^[\S+]+) - - \[(.*)\]␣
 ↪"(\w+) (.*?) (.*?)', 4).alias('endpoint'),
                        regexp_extract('value', r'(^[\S+]+) - - \[(.*) \+(.
 ↪*)\] "(\w+) (.*?) (.*?)" ', 6).alias('protocol'),
                        regexp_extract('value', r'(^[\S+]+) - - \[(.*) \+(.
 ↪*)\] "(\w+) (.*?) (.*?)" (\d+) ', 7).cast('integer').alias('status'),
```

```
                           regexp_extract('value', r'(^[\S+]+) - - \[(.*) \+(.
 ↪*)\] "(\w+) (.*?) (.*?)" (\d+) (\d+)', 8).cast('integer').
 ↪alias('content_size'))
log_df.show(10, truncate=True)
print((log_df.count(), len(log_df.columns)))
```

```
+--------------------+--------------------+------+---------+--------+------+----
--------+
|                host|           timestamp|method|
endpoint|protocol|status|content_size|
+--------------------+--------------------+------+---------+--------+------+----
--------+
|2001:6d8:10:4400:…|10/Nov/2019:03:44…|   GET|/wpad.dat|HTTP/1.1|   304|
null|
|2001:6d8:10:4400:…|10/Nov/2019:03:44…|   GET|/wpad.dat|HTTP/1.1|   200|
137|
|       172.30.254.52|10/Nov/2019:03:44…|   GET|/wpad.dat|HTTP/1.1|   200|
137|
|2001:6d8:10:4400:…|10/Nov/2019:03:44…|   GET|/wpad.dat|HTTP/1.1|   200|
137|
|       172.30.254.52|10/Nov/2019:03:44…|   GET|/wpad.dat|HTTP/1.1|   200|
137|
|2001:6d8:10:4401:…|10/Nov/2019:03:44…|   GET|/wpad.dat|HTTP/1.1|   304|
null|
|2001:6d8:10:4400:…|10/Nov/2019:03:44…|   GET|/wpad.dat|HTTP/1.1|   200|
137|
|       172.30.254.52|10/Nov/2019:03:44…|   GET|/wpad.dat|HTTP/1.1|   200|
137|
|2001:6d8:10:4400:…|10/Nov/2019:03:44…|   GET|/wpad.dat|HTTP/1.1|   304|
null|
|2001:6d8:10:4400:…|10/Nov/2019:03:44…|   GET|/wpad.dat|HTTP/1.1|   200|
137|
+--------------------+--------------------+------+---------+--------+------+----
--------+
only showing top 10 rows

(285061, 7)
```

[9]:
```
(log_df
    .filter(log_df['content_size']
                .isNull())
    .count())
```

[9]: 175714

[10]:
```
bad_rows_df = log_df.filter(log_df['host'].isNull()|
                            log_df['timestamp'].isNull() |
```

```
                            log_df['method'].isNull() |
                            log_df['endpoint'].isNull() |
                            log_df['status'].isNull() |
                            log_df['content_size'].isNull()|
                            log_df['protocol'].isNull())
bad_rows_df.count()
```

[10]:
```
175714
```

[11]:
```
bad_rows_df.show(5)
```

```
+--------------------+--------------------+------+--------+--------+------+----
--------+
|                host|           timestamp|method|
endpoint|protocol|status|content_size|
+--------------------+--------------------+------+--------+--------+------+----
--------+
|2001:6d8:10:4400:…|10/Nov/2019:03:44…|   GET|/wpad.dat|HTTP/1.1|   304|
null|
|2001:6d8:10:4401:…|10/Nov/2019:03:44…|   GET|/wpad.dat|HTTP/1.1|   304|
null|
|2001:6d8:10:4400:…|10/Nov/2019:03:44…|   GET|/wpad.dat|HTTP/1.1|   304|
null|
|2001:6d8:10:4400:…|10/Nov/2019:03:44…|   GET|/wpad.dat|HTTP/1.1|   304|
null|
|2001:6d8:10:4400:…|10/Nov/2019:03:44…|   GET|/wpad.dat|HTTP/1.1|   304|
null|
+--------------------+--------------------+------+--------+--------+------+----
--------+
only showing top 5 rows
```

[12]:
```
log_df = log_df.na.fill({'content_size': 0})
log_df.count()
```

[12]:
```
285061
```

[13]:
```
log_df1 = log_df[~log_df['timestamp'].isin([''])]
log_df1.count()
```

[13]:
```
283386
```

[15]:
```
from pyspark.sql.functions import udf

month_map = {
  'Jan': 1, 'Feb': 2, 'Mar':3, 'Apr':4, 'May':5, 'Jun':6, 'Jul':7,
  'Aug':8,  'Sep': 9, 'Oct':10, 'Nov': 11, 'Dec': 12
```

```
}

def parse_clf_time(text):
    """ Convert Common Log time format into a Python datetime object
    Args:
        text (str): date and time in Apache time format [dd/mmm/yyyy:hh:mm:ss␣
 ↪(+/-)zzzz]
    Returns:
        a string suitable for passing to CAST('timestamp')
    """
    # NOTE: We're ignoring the time zones here, might need to be handled␣
 ↪depending on the problem you are solving
    return "{0:04d}-{1:02d}-{2:02d} {3:02d}:{4:02d}:{5:02d}".format(
      int(text[7:11]),
      month_map[text[3:6]],
      int(text[0:2]),
      int(text[12:14]),
      int(text[15:17]),
      int(text[18:20])
    )
```

[16]:
```
udf_parse_time = udf(parse_clf_time)

log_dfn = (log_df1.select('*', udf_parse_time(log_df1['timestamp'])
                                    .cast('timestamp')
                                    .alias('time'))
                                    .drop('timestamp'))
```

[17]:
```
log_dfn.printSchema()
```

```
root
 |-- host: string (nullable = true)
 |-- method: string (nullable = true)
 |-- endpoint: string (nullable = true)
 |-- protocol: string (nullable = true)
 |-- status: integer (nullable = true)
 |-- content_size: integer (nullable = false)
 |-- time: timestamp (nullable = true)
```

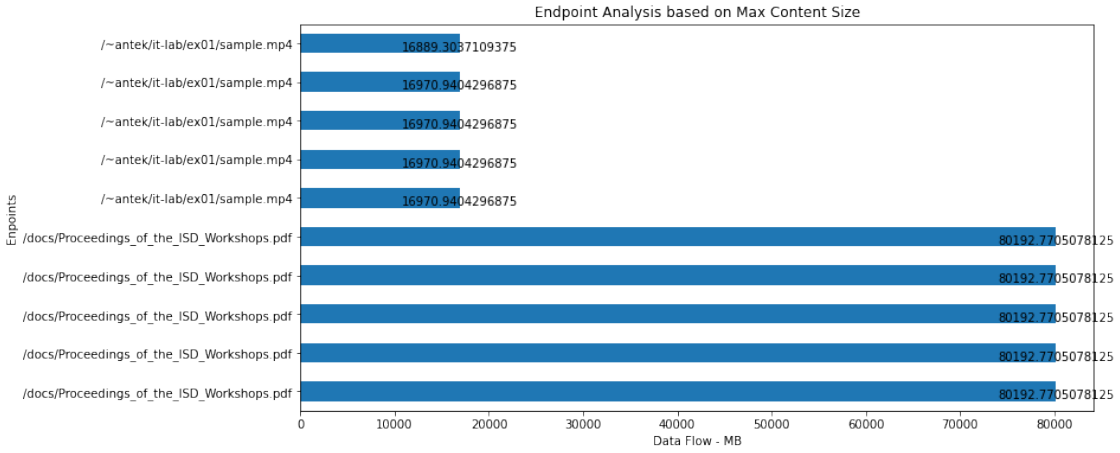
[18]:
```
log_dfn.createOrReplaceTempView("logs")
```

[19]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
%matplotlib inline
```

[20]:
```
def bar_plot_list_of_tuples_horizontal(input_list,x_label,y_label,plot_title):
    y_labels = [val[0] for val in input_list]
    x_labels = [val[1] for val in input_list]
    plt.figure(figsize=(12, 6))
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    plt.title(plot_title)
    ax = pd.Series(x_labels).plot(kind='barh')
    ax.set_yticklabels(y_labels)
    for i, v in enumerate(x_labels):
        ax.text(int(v) + 0.5, i - 0.25, str(v),ha='center', va='bottom')
```

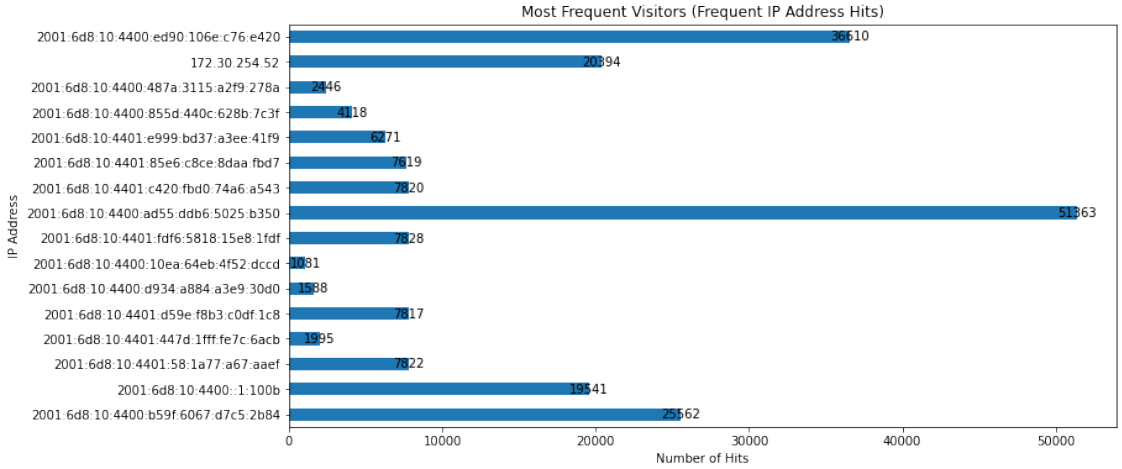
[21]:
```
topEndpointsMaxSize = (sqlContext
                .sql("SELECT endpoint,content_size/1024 FROM logs ORDER BY␣
 ↪content_size DESC LIMIT 10")
                .rdd.map(lambda row: (row[0], row[1]))
                .collect())
```

[22]:
```
bar_plot_list_of_tuples_horizontal(topEndpointsMaxSize,'Data Flow -␣
 ↪MB','Enpoints','Endpoint Analysis based on Max Content Size')
```



[23]:
```
frequentIpAddressesHits = (sqlContext
               .sql("SELECT host, COUNT(*) AS total FROM logs GROUP BY host␣
 ↪HAVING total > 1000")
               .rdd.map(lambda row: (row[0], row[1]))
               .collect())
```

[24]:
```
bar_plot_list_of_tuples_horizontal(frequentIpAddressesHits,'Number of Hits','IP␣
 ↪Address','Most Frequent Visitors (Frequent IP Address Hits)')
```



[25]:
```
import pyspark.sql.functions as F
host_day_df = log_dfn.select(log_dfn.host,
                             F.dayofmonth('time').alias('day'))
```

[26]:
```
status_freq_df = (log_dfn
                      .groupBy('status')
                      .count()
                      .sort('status')
                      .cache())
print('Total distinct HTTP Status Codes:', status_freq_df.count())
```

```
Total distinct HTTP Status Codes: 10
```

[27]:
```
log_freq_df = status_freq_df.withColumn('log(count)',
                                        F.log(status_freq_df['count']))
log_freq_df.show()
```

```
+------+------+------------------+
|status| count|        log(count)|
+------+------+------------------+
|  null|   393| 5.973809611869261|
|   200|108149|11.591265184921443|
|   206|    65| 4.174387269895637|
|   301|    52|3.9512437185814275|
|   302|    33|3.4965075614664802|
|   304|172698|12.059299683291801|
|   400|     1|               0.0|
|   401|   508| 6.230481447578482|
|   403|    37|3.6109179126442243|
|   404|  1450|  7.27931883541462|
+------+------+------------------+
```

[28]:
```
host_day_df.show(5, truncate=False)
```

```
+----------------------------------+---+
|host                              |day|
+----------------------------------+---+
|2001:6d8:10:4400:451:aebb:715b:b6df|10 |
|2001:6d8:10:4400::1:100b          |10 |
|172.30.254.52                     |10 |
|2001:6d8:10:4400::1:100b          |10 |
|172.30.254.52                     |10 |
+----------------------------------+---+
only showing top 5 rows
```

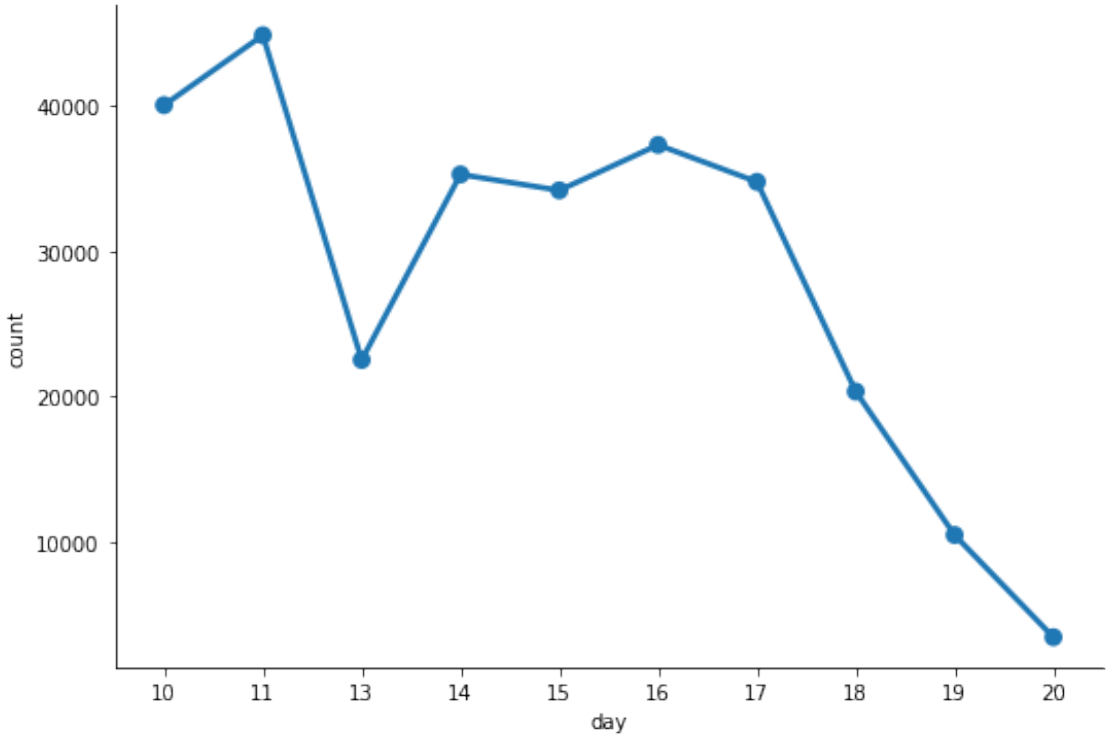
[29]:
```
def_mr = pd.get_option('max_rows')
pd.set_option('max_rows', 10)

daily_hosts_df = (host_day_df
                      .groupBy('day')
                      .count()
                      .sort("day"))
```

[30]:
```
daily_hosts_df.show(10, truncate=True)
```

```
+---+-----+
|day|count|
+---+-----+
| 10|40070|
| 11|44879|
| 13|22531|
| 14|35302|
| 15|34202|
| 16|37332|
| 17|34789|
| 18|20355|
| 19|10480|
| 20| 3446|
+---+-----+
```

[31]:
```
daily_hosts_pd_df = (daily_hosts_df
                         .toPandas()
                         .sort_values(by=['count'],
                                      ascending=False))
```

[32]:
```
c = sns.catplot(x='day', y='count',
                data=daily_hosts_pd_df,
                kind='point', height=5,
                aspect=1.5)
```



[33]:
```
from pyspark.sql import SparkSession
spark = SparkSession \
        .builder \
        .appName("Python Spark Data Exploration") \
        .config("spark.some.config.option", "some-value") \
        .getOrCreate()
```

[34]:
```
#import libraries
import pandas as pd
import matplotlib.pyplot as plt
import random

%matplotlib inline

#set ggplot style
plt.style.use('ggplot')
```

[35]:
```
df = spark.read.format('com.databricks.spark.csv').\
                                options(header='true', \
                                inferschema='true').load("/home/spark/files/2008.
↪csv",header=True);
```

[36]:
```
df.columns
```

[36]:
```
['Year',
 'Month',
 'DayofMonth',
 'DayOfWeek',
 'DepTime',
 'CRSDepTime',
 'ArrTime',
 'CRSArrTime',
 'UniqueCarrier',
 'FlightNum',
 'TailNum',
 'ActualElapsedTime',
 'CRSElapsedTime',
 'AirTime',
 'ArrDelay',
 'DepDelay',
 'Origin',
 'Dest',
 'Distance',
 'TaxiIn',
 'TaxiOut',
 'Cancelled',
 'CancellationCode',
 'Diverted',
 'CarrierDelay',
 'WeatherDelay',
 'NASDelay',
 'SecurityDelay',
 'LateAircraftDelay']
```

[37]:
```
df.groupBy("UniqueCarrier").count().show()
```

```
+-------------+-------+
|UniqueCarrier|  count|
+-------------+-------+
|           UA| 449515|
|           AA| 604885|
|           NW| 347652|
|           EV| 280575|
|           B6| 196091|
```

```
|           DL| 451931|
|           OO| 567159|
|           F9|  95762|
|           YV| 254930|
|           US| 453589|
|           AQ|   7800|
|           MQ| 490693|
|           OH| 197607|
|           HA|  61826|
|           XE| 374510|
|           AS| 151102|
|           FL| 261684|
|           CO| 298455|
|           WN|1201754|
|           9E| 262208|
+-------------+-------+
```

[38]:
```
df.cache
```

[38]:
```
<bound method DataFrame.cache of DataFrame[Year: int, Month: int, DayofMonth:
int, DayOfWeek: int, DepTime: string, CRSDepTime: int, ArrTime: string,
CRSArrTime: int, UniqueCarrier: string, FlightNum: int, TailNum: string,
ActualElapsedTime: string, CRSElapsedTime: string, AirTime: string, ArrDelay:
string, DepDelay: string, Origin: string, Dest: string, Distance: int, TaxiIn:
string, TaxiOut: string, Cancelled: int, CancellationCode: string, Diverted:
int, CarrierDelay: string, WeatherDelay: string, NASDelay: string,
SecurityDelay: string, LateAircraftDelay: string]>
```

[39]:
```
df.createOrReplaceTempView("flights")
```

[40]:
```
spark.catalog.cacheTable("flights")
```

[94]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
%matplotlib inline
```

[95]:
```
import pyspark.sql.functions as F
crshour = df.select('DepDelay', F.round(F.col('CRSDepTime')/100).
 ↪cast('integer').alias('CRSDepHour'))
```

[96]:
```
crshour.show()
```

```
+--------+----------+
|DepDelay|CRSDepHour|
+--------+----------+
|       8|        20|
```

```
|      19|         7|
|       8|         6|
|      -4|         9|
|      34|        18|
|      25|        19|
|      67|        18|
|      -1|        10|
|       2|         6|
|       0|        16|
|       6|         7|
|      94|        15|
|      -4|        14|
|       0|         7|
|       2|        17|
|       9|        10|
|      27|        14|
|       9|         7|
|      28|        13|
|      51|        13|
+--------+----------+
only showing top 20 rows
```

[97]:
```
crshour_count = ( crshour.filter(df.DepDelay > 20).groupBy('crsdephour').
↪count().sort('crsdephour').cache())
```

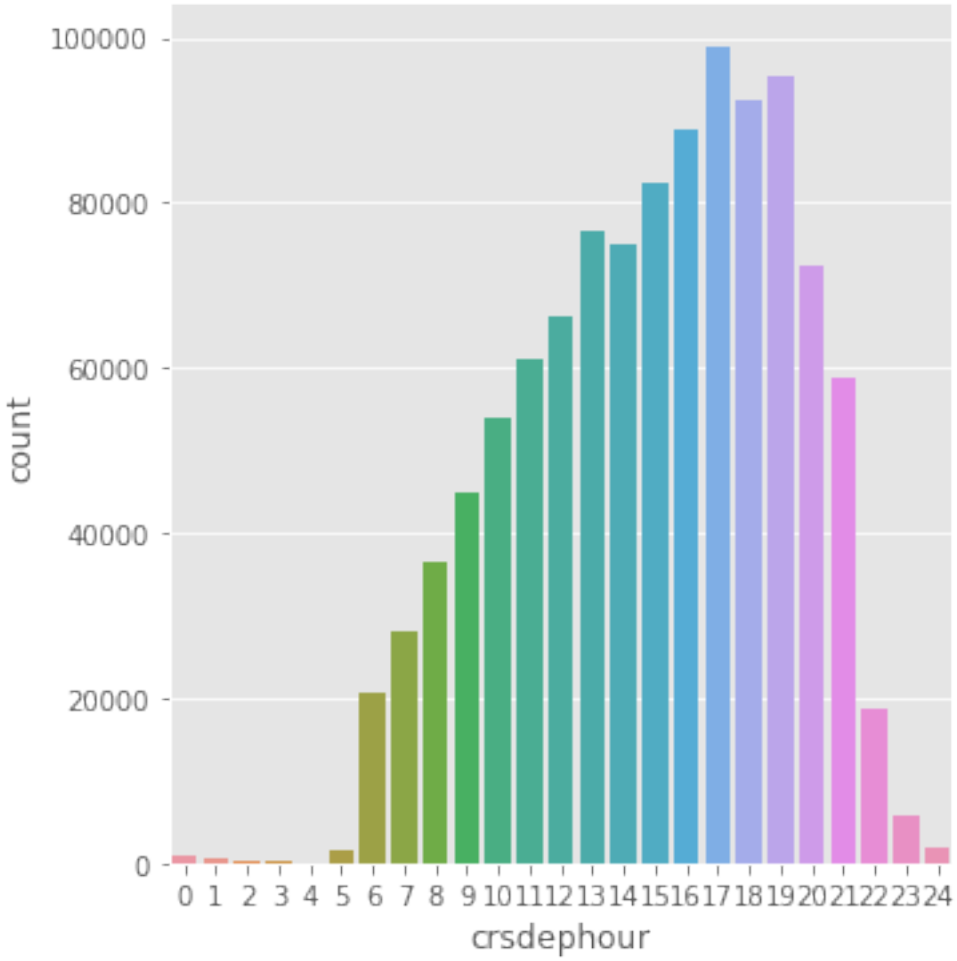
[98]:
```
crshour_count_pd = ( crshour_count.toPandas().sort_values(by=['crsdephour']))
```

[99]:
```
crshour_count_pd
```

[99]:
```
    crsdephour  count
0            0    798
1            1    558
2            2    125
3            3    106
4            4     88
..         …    …
20          20  72519
21          21  58941
22          22  18554
23          23   5861
24          24   1925

[25 rows x 2 columns]
```

[100]:
```
sns.catplot(x='crsdephour', y='count',␣
↪data=crshour_count_pd,kind='bar',order=crshour_count_pd['crsdephour'] )
```

[100]: <seaborn.axisgrid.FacetGrid at 0x7f86462a55b0>



[41]:
```
#ZADANIE DOMOWE
```

[42]:
```
#ZAD1 Przedstawić tabelę zawierającą opóźnienia powyżej 40 minut na lotnisku␣
↪docelowym.
#Wykorzystać transformacje: filter(), groupBy(), count(), orderBy().
```

[74]:
```
df.filter(df.DepDelay > 40).groupBy("Dest").count().orderBy(['count'],␣
↪ascending = [False]).show(10)
```

```
+----+-----+
|Dest|count|
+----+-----+
| ORD|49602|
| ATL|35715|
| EWR|27951|
```

```
| DFW|22963|
| SFO|19658|
| DEN|18294|
| LAX|16799|
| LGA|15978|
| JFK|14827|
| LAS|14351|
+----+-----+
only showing top 10 rows
```

[71]:
```
#ZAD2 Przedstawić tabelę zawierającą informację o przewoźniku, lotnisku␣
 ↪startowym, docelowym,
#czasie opóźnienia opóźnienia i czasie lądowania dla lotów z opóźnieniem␣
 ↪powyżej 40 minut na lotnisku docelowym.
#Do realizacji zapytania wykorzystać pytanie SQL do utworzonego widoku lub␣
 ↪polecenie select na DF.
```

[106]:
```
df.createGlobalTempView("carriers1")
spark.sql("SELECT UniqueCarrier, Origin, Dest, DepDelay, CRSDepTime \
    from global_temp.carriers1 where DepDelay > 40 order By DepDelay Desc, dest␣
 ↪asc").show(5)
```

```
+-------------+------+----+--------+----------+
|UniqueCarrier|Origin|Dest|DepDelay|CRSDepTime|
+-------------+------+----+--------+----------+
|           NW|   LAX| MSP|     999|       810|
|           AA|   BDL| DFW|     998|      1325|
|           NW|   MEM| MSP|     997|      1345|
|           NW|   DTW| CMH|     996|      1523|
|           OO|   RFD| DEN|     996|       600|
+-------------+------+----+--------+----------+
only showing top 5 rows
```

[70]:
```
#ZAD3 Wyznaczyć średnie opóźnienia dla poszczególnych przewoźników.
#Realizacja zadania na DF - transformacje groupBy() i agg().
```

[93]:
```
df.groupBy("UniqueCarrier").agg({'DepDelay': 'mean' }).show()
```

```
+-------------+------------------+
|UniqueCarrier|      avg(DepDelay)|
+-------------+------------------+
|           UA|  14.11257661236138|
|           AA| 13.280898264437912|
|           NW|  6.463235656670833|
|           EV| 11.922537970871462|
|           B6| 12.653395748122113|
```

```
|           DL|  8.007765572702564|
|           OO| 7.4564427592619955|
|           F9|  5.919601516833923|
|           YV| 12.000675279875033|
|           US|  5.717489671893907|
|           AQ|-1.3977829337458108|
|           MQ| 10.695641776641581|
|           OH| 11.536153117856601|
|           HA| 0.4552013450206487|
|           XE| 11.395866476493499|
|           AS|  6.848722010417226|
|           FL|  9.262713040260852|
|           CO|  13.18522978602152|
|           WN| 10.383034750411133|
|           9E|  6.765859659983622|
+-------------+-------------------+
```

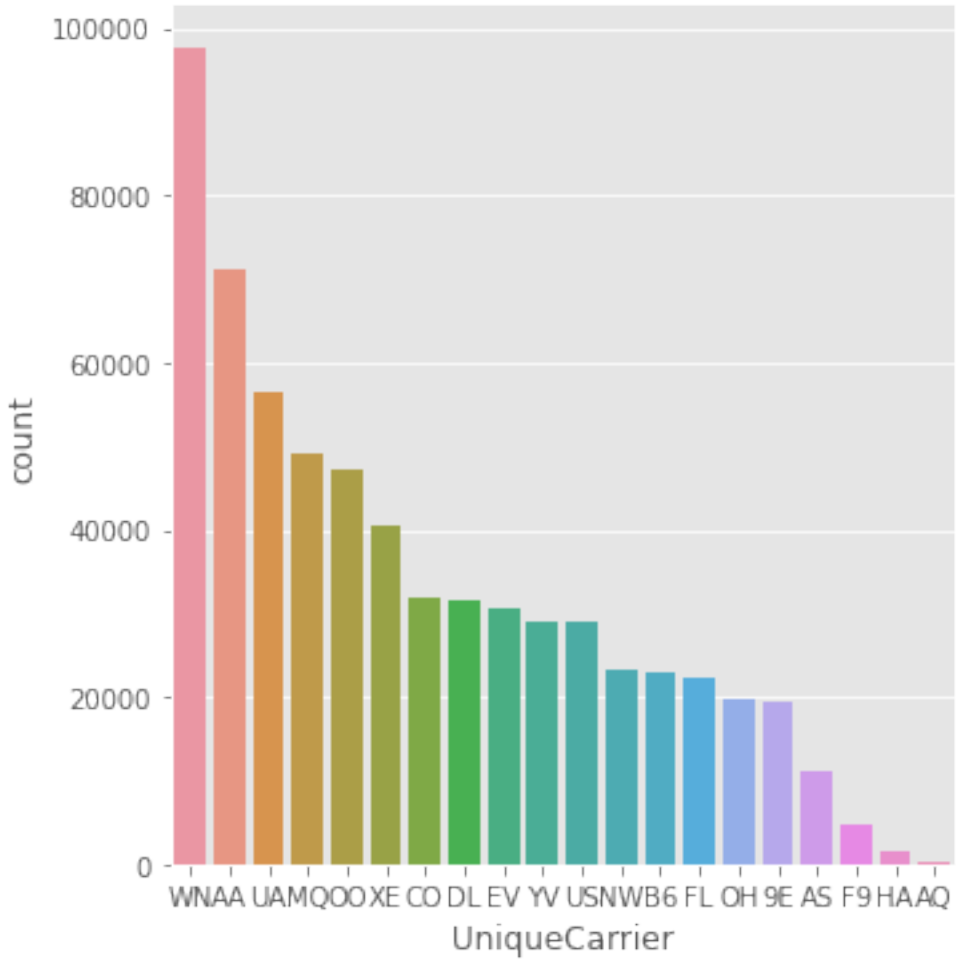
[72]:
```
#ZAD4 Przedstawić tabelę oraz wykres prezentujący liczbę opóźnień dla␣
↪przewoźników (powyżej 40 minut ).
```

[187]:
```
crshour = df.select('UniqueCarrier','DepDelay')
crshour_count = ( crshour.filter(df.DepDelay > 40).groupBy('UniqueCarrier').
↪count().sort('UniqueCarrier').cache())
crshour_count_pd = ( crshour_count.toPandas().sort_values(by=['count'],␣
↪ascending=False))
crshour_count_pd
```

[187]:
```
   UniqueCarrier  count
17            WN  97761
1             AA  71041
15            UA  56312
11            MQ  49203
14            OO  47291
..           …    …
0             9E  19345
3             AS  11031
8             F9   4881
10            HA   1393
2             AQ    128

[20 rows x 2 columns]
```

[188]:
```
sns.catplot(x='UniqueCarrier', y='count',␣
↪data=crshour_count_pd,kind='bar',order=crshour_count_pd['UniqueCarrier'] )
```

[188]:
```
<seaborn.axisgrid.FacetGrid at 0x7f863b8a7550>
```



[73]:
```
#ZAD5 Przedstawić tabelę oraz wykres prezentujący liczbę opóźnień ( powyżej 20
↪minut ) w zależności od dnia tygodnia.
```

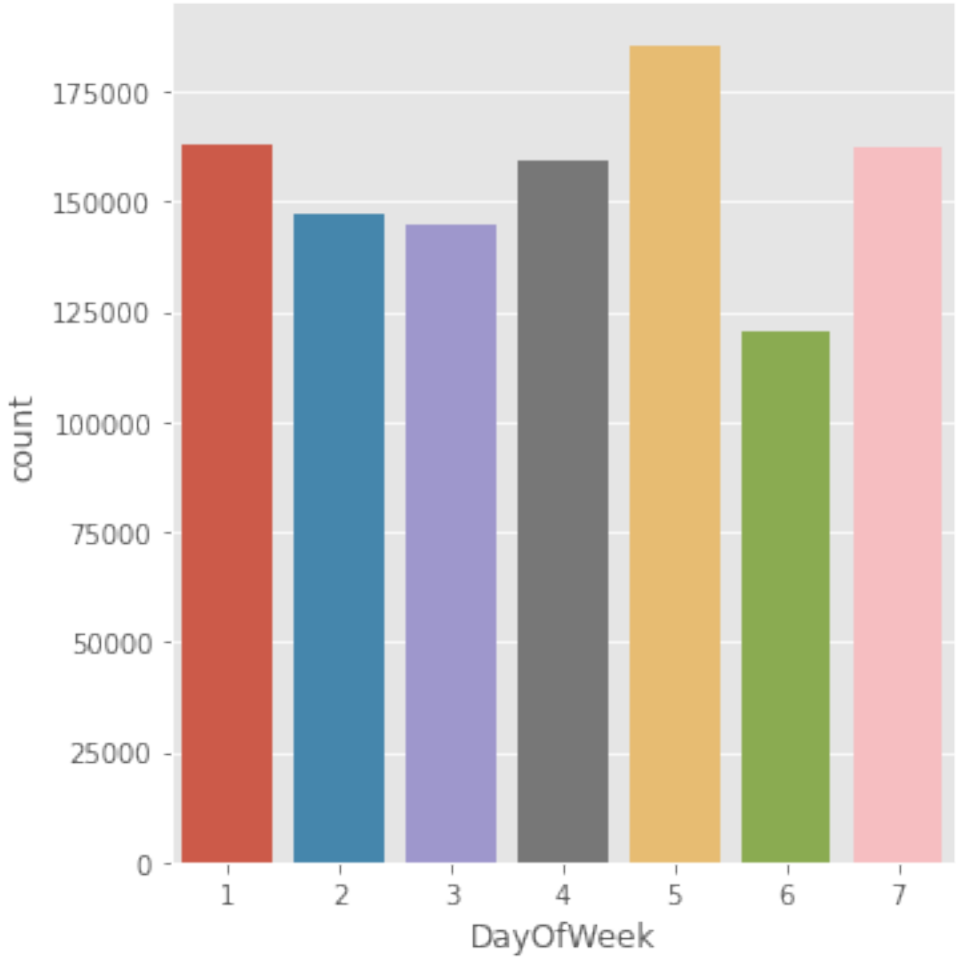
[151]:
```
crshour = df.select('DepDelay', 'DayOfWeek')
crshour_count = ( crshour.filter(df.DepDelay > 20).groupBy('DayOfWeek').count().
↪sort('DayOfWeek').cache())
crshour_count_pd = ( crshour_count.toPandas().sort_values(by=['DayOfWeek']))
crshour_count_pd
```

[151]:

	DayOfWeek	count
0	1	162834
1	2	147174
2	3	144570
3	4	159197
4	5	185572
5	6	120555
6	7	162469

[152]:
```
import seaborn as sns
sns.catplot(x='DayOfWeek', y='count',␣
 ↪data=crshour_count_pd,kind='bar',order=crshour_count_pd['DayOfWeek'] )
```

[152]: <seaborn.axisgrid.FacetGrid at 0x7f863be14a30>



[ ]: