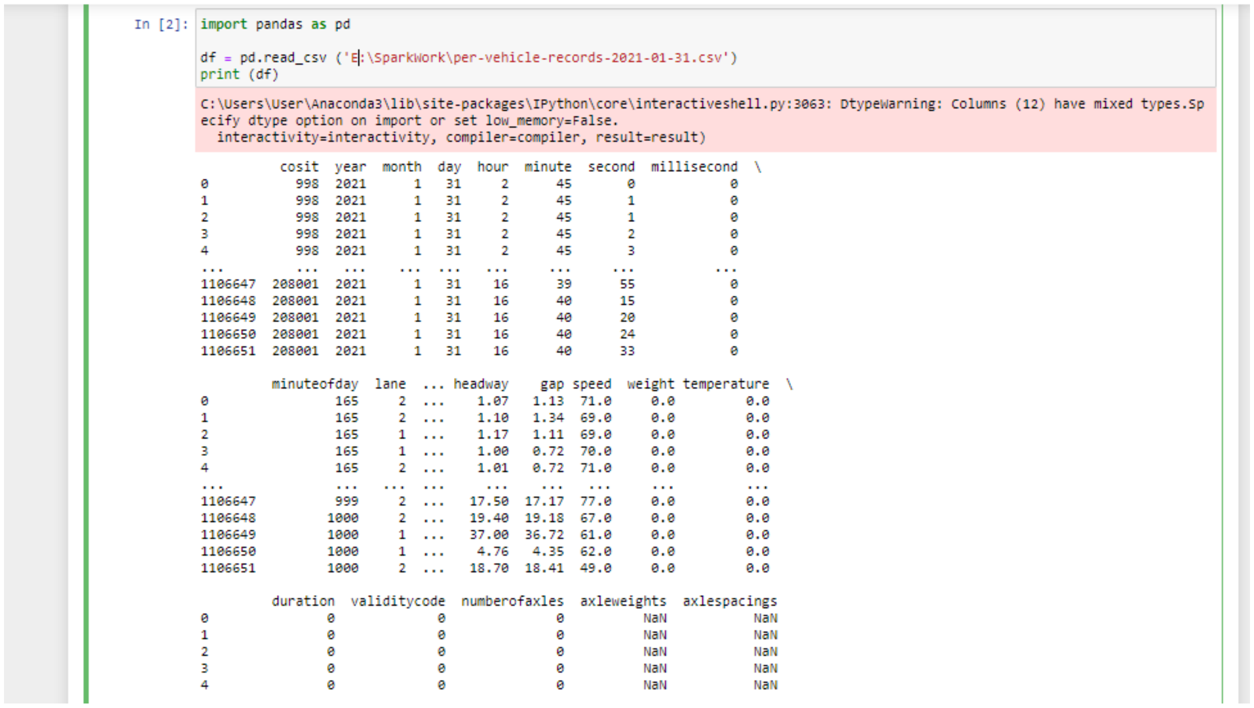
SUMMARY ANALYTICS

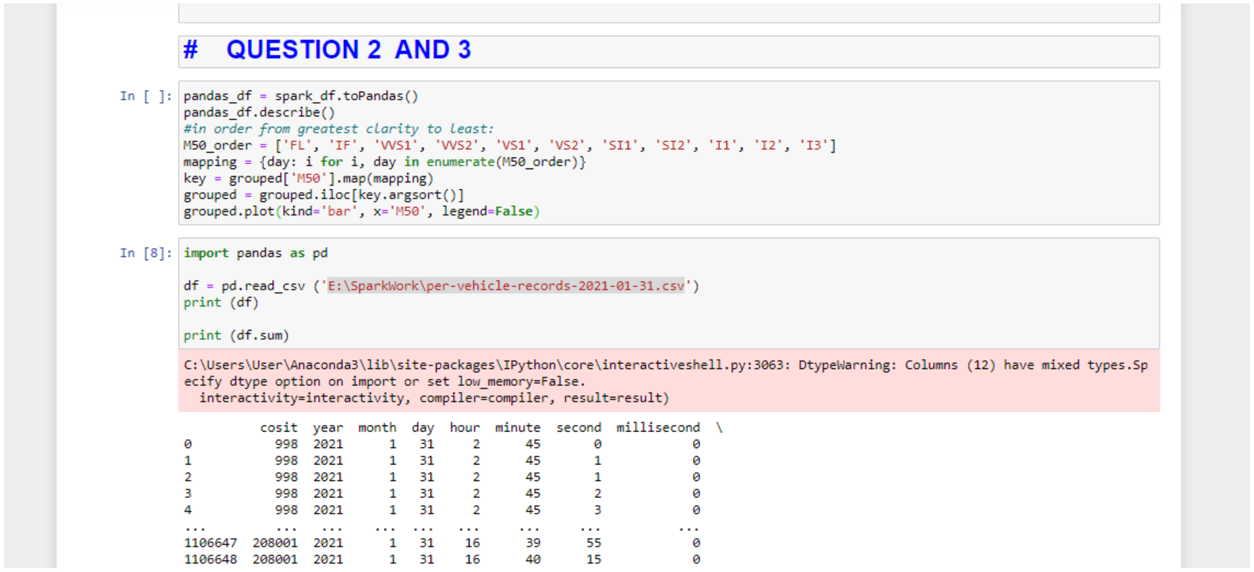
.

QUESTION 1



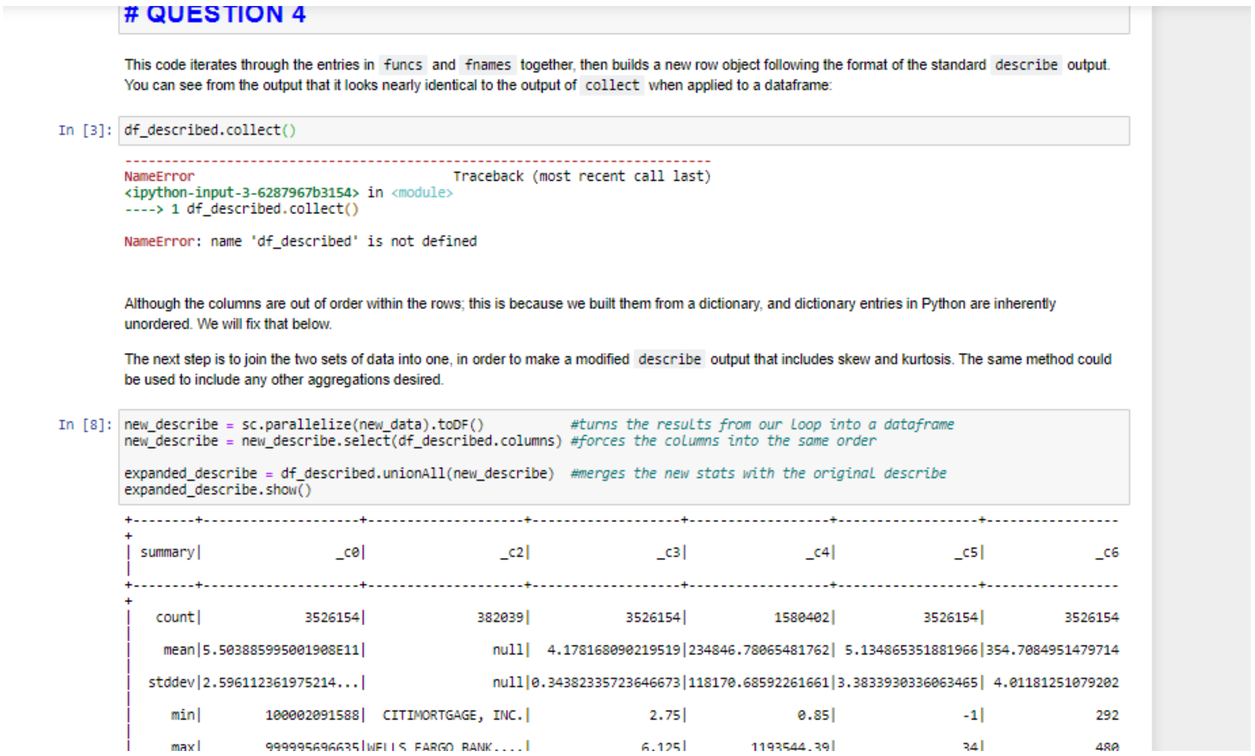
```
In [2]: import pandas as pd

        df = pd.read_csv ('E:\SparkWork\per-vehicle-records-2021-01-31.csv')
        print (df)

C:\Users\User\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3063: DtypeWarning: Columns (12) have mixed types.Sp
ecify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)

          cosit  year  month  day  hour  minute  second  millisecond  \
0           998  2021      1   31     2      45       0            0
1           998  2021      1   31     2      45       1            0
2           998  2021      1   31     2      45       1            0
3           998  2021      1   31     2      45       2            0
4           998  2021      1   31     2      45       3            0
...         ...   ...    ...  ...   ...     ...     ...          ...
1106647  208001  2021      1   31    16      39      55            0
1106648  208001  2021      1   31    16      40      15            0
1106649  208001  2021      1   31    16      40      20            0
1106650  208001  2021      1   31    16      40      24            0
1106651  208001  2021      1   31    16      40      33            0

         minuteofday  lane  ... headway    gap speed  weight temperature  \
0                165     2  ...    1.07   1.13  71.0     0.0         0.0
1                165     2  ...    1.10   1.34  69.0     0.0         0.0
2                165     1  ...    1.17   1.11  69.0     0.0         0.0
3                165     1  ...    1.00   0.72  70.0     0.0         0.0
4                165     2  ...    1.01   0.72  71.0     0.0         0.0
...              ...   ...  ...     ...    ...   ...     ...         ...
1106647          999     2  ...   17.50  17.17  77.0     0.0         0.0
1106648         1000     2  ...   19.40  19.18  67.0     0.0         0.0
1106649         1000     1  ...   37.00  36.72  61.0     0.0         0.0
1106650         1000     1  ...    4.76   4.35  62.0     0.0         0.0
1106651         1000     2  ...   18.70  18.41  49.0     0.0         0.0

         duration  validitycode  numberofaxles  axleweights  axlespacings
0               0             0              0          NaN           NaN
1               0             0              0          NaN           NaN
2               0             0              0          NaN           NaN
3               0             0              0          NaN           NaN
4               0             0              0          NaN           NaN
```

QUESTION 2 AND 3



```
#   QUESTION 2  AND 3

In [ ]: pandas_df = spark_df.toPandas()
        pandas_df.describe()
        #in order from greatest clarity to least:
        M50_order = ['FL', 'IF', 'VVS1', 'VVS2', 'VS1', 'VS2', 'SI1', 'SI2', 'I1', 'I2', 'I3']
        mapping = {day: i for i, day in enumerate(M50_order)}
        key = grouped['M50'].map(mapping)
        grouped = grouped.iloc[key.argsort()]
        grouped.plot(kind='bar', x='M50', legend=False)

In [8]: import pandas as pd

        df = pd.read_csv ('E:\SparkWork\per-vehicle-records-2021-01-31.csv')
        print (df)

        print (df.sum)

C:\Users\User\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3063: DtypeWarning: Columns (12) have mixed types.Sp
ecify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)

          cosit  year  month  day  hour  minute  second  millisecond  \
0           998  2021      1   31     2      45       0            0
1           998  2021      1   31     2      45       1            0
2           998  2021      1   31     2      45       1            0
3           998  2021      1   31     2      45       2            0
4           998  2021      1   31     2      45       3            0
...         ...   ...    ...  ...   ...     ...     ...          ...
1106647  208001  2021      1   31    16      39      55            0
1106648  208001  2021      1   31    16      40      15            0
```

QUESTION 4



QUESTION 4

This code iterates through the entries in `funcs` and `fnames` together, then builds a new row object following the format of the standard `describe` output. You can see from the output that it looks nearly identical to the output of `collect` when applied to a dataframe:

```
In [3]: df_described.collect()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-6287967b3154> in <module>
----> 1 df_described.collect()

NameError: name 'df_described' is not defined
```

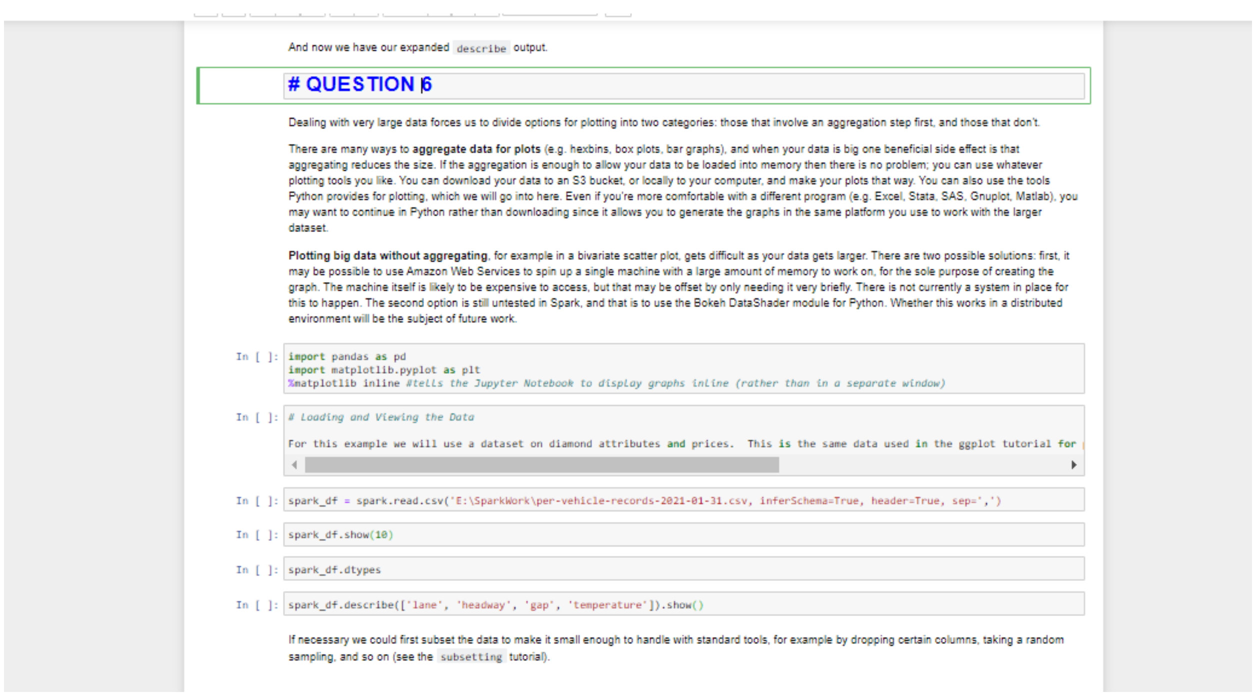
Although the columns are out of order within the rows; this is because we built them from a dictionary, and dictionary entries in Python are inherently unordered. We will fix that below.

The next step is to join the two sets of data into one, in order to make a modified `describe` output that includes skew and kurtosis. The same method could be used to include any other aggregations desired.

```
In [8]: new_describe = sc.parallelize(new_data).toDF()          #turns the results from our loop into a dataframe
        new_describe = new_describe.select(df_described.columns) #forces the columns into the same order

        expanded_describe = df_described.unionAll(new_describe)  #merges the new stats with the original describe
        expanded_describe.show()
```

```
+--------+-------------------+-------------------+------------------+-----------------+-----------------+----------------
+
| summary|                _c0|                _c2|               _c3|              _c4|              _c5|             _c6
|
+--------+-------------------+-------------------+------------------+-----------------+-----------------+----------------
+
|   count|            3526154|             382039|           3526154|          1580402|          3526154|         3526154
|
|    mean|5.503885995001908E11|               null|  4.178168090219519|234846.78065481762| 5.134865351881966|354.7084951479714
|
|  stddev|2.596112361975214...|               null|0.3438233572364667 3|118170.68592261661|3.3833930336063465| 4.01181251079202
|
|     min|        100002091588|  CITIMORTGAGE, INC.|              2.75|             0.85|               -1|             292
|
|     max|        999995696635|WELLS FARGO BANK....|             6.125|       1193544.39|               34|             480
```

QUESTION 5 AND 6


And now we have our expanded `describe` output.

QUESTION 6

Dealing with very large data forces us to divide options for plotting into two categories: those that involve an aggregation step first, and those that don't.

There are many ways to **aggregate data for plots** (e.g. hexbins, box plots, bar graphs), and when your data is big one beneficial side effect is that aggregating reduces the size. If the aggregation is enough to allow your data to be loaded into memory then there is no problem; you can use whatever plotting tools you like. You can download your data to an S3 bucket, or locally to your computer, and make your plots that way. You can also use the tools Python provides for plotting, which we will go into here. Even if you're more comfortable with a different program (e.g. Excel, Stata, SAS, Gnuplot, Matlab), you may want to continue in Python rather than downloading since it allows you to generate the graphs in the same platform you use to work with the larger dataset.

Plotting big data without aggregating, for example in a bivariate scatter plot, gets difficult as your data gets larger. There are two possible solutions: first, it may be possible to use Amazon Web Services to spin up a single machine with a large amount of memory to work on, for the sole purpose of creating the graph. The machine itself is likely to be expensive to access, but that may be offset by only needing it very briefly. There is not currently a system in place for this to happen. The second option is still untested in Spark, and that is to use the Bokeh DataShader module for Python. Whether this works in a distributed environment will be the subject of future work.

```
In [ ]: import pandas as pd
        import matplotlib.pyplot as plt
        %matplotlib inline #tells the Jupyter Notebook to display graphs inline (rather than in a separate window)
```

```
In [ ]: # Loading and Viewing the Data

        For this example we will use a dataset on diamond attributes and prices.  This is the same data used in the ggplot tutorial for
```

```
In [ ]: spark_df = spark.read.csv('E:\SparkWork\per-vehicle-records-2021-01-31.csv, inferSchema=True, header=True, sep=',')
```

```
In [ ]: spark_df.show(10)
```

```
In [ ]: spark_df.dtypes
```

```
In [ ]: spark_df.describe(['lane', 'headway', 'gap', 'temperature']).show()
```

If necessary we could first subset the data to make it small enough to handle with standard tools, for example by dropping certain columns, taking a random sampling, and so on (see the `subsetting` tutorial).

CODE AND OUTPUT

QUESTION 1

```
tot = vehicle_counter_DF. count( )

print ( "Total Vehicle Entry print ( tot )

 groupBy( " classname " ) \

. count( ) \

.withC01umnRenamed( ' count ' , ' Count' ) \

              . withC01umn( ' Percentage '      (F.col( 'Count '

. show( )                           tot )
```

In [63] :

Total Vehicle Entry 1106652

QUESTION 2

I classnamel Count I Percentage I

CARI 918254 | 82 .97585871619985 1 HGV_ARTI 33805 1 3. 05470915879608 1 BUSI 10519 1 0 . 9505246455073502 1 HGV_RIGI 308661 2 . 7891333499600597 | null I 50 1 0 . 004518132168016684 1 CARAVAN 1 5887 1 0 .5319648814622845 1

LGVI 104580 | 9 .450125242623697 1

MBIKE I 2691 1 0 .24316587328265796 1

Question 3—5

# Calculate the highest and lowest hourly fows on M50 — from pyspark . sql import Window

ExampleDF = M50DF . groupBy("hour") \

. count( ) \

.withC01umnRenamed(' count' , 'Total Vehicle Count ')

print("Lowest Hourly Flow") resDF = ExampleDF . filter(col("Total Vehicle Count resDF . show( )

print("Highest Hourly Flow") resDF = ExampleDF . filter(col("Total Vehicle Count resDF . show( )show the hours and total number of vehicle counts.

ExampleDF . groupby( ) . min(' Total Vehicle Count') . head( )

ExampleDF . groupby( ) . max(' Total Vehicle Count') . head( )

In [167] :

Lowest Hourly Flow

I hour I Total Vehicle Count I

3 1 510 1

Highest Hourly Flow

I hour I Total Vehicle Count I

15 1 172111

I have assumed : Morning Hours —> 8 to 11 and Evening Hours 17 to 20 morningDF = M50DF.fi1ter(c01("hour") '8') . "hour") <= ' 11') . orderBy(" hour")

morningRushDF = morningDF . groupBy(" hour"

. count( ) \

. withC01umnRenamed (count '' Total Vehicle Count')

print("Morning Rush Hour") resDF = morningRushDF . filter (col("Total Vehicle Count ") morningRushDF . groupby( ) . max('Total Vehicle Count ') . head( ) [0 resDF . show( )

eveningDF = M50DF.fi1ter(c01("hour") >= ' 17 ') . "hour") '20 ') . orderBy(" hour")

eveningRushDF = eveningDF . groupBy(" hour") \

. count( ) \

.withC01umnRenamed(' count' , 'Total Vehicle Count ')

print("Evening Rush Hours ") resDF = eveningRushDF . filter (col("Total Vehicle Count " = eveningRushDF. groupby( ) . max('Total Vehicle Count ') . head( ) [0 resDF . show( )

Morning Rush Hour

I hour I Total Vehicle Count I

Evening Rush Hours

I hour I Total Vehicle Count I

```
jun14 = M50DF.fi1ter(c01( "cosit") 15010) . orderBy( "hour" ) jun14 . groupBy( ) . agg(F . sum( " speed" ) .
alias( "count " ) ) \

            . withC01umnRenamed( ' count ' ,          ' totalSpeed' ) \

            . withC01umn( ' Average Speed '            (F. col( ' total Speed' ) / total Speed) )

. show( )

print( "Avg Speed between Junction 15 and 16 " jun15 = M50DF.  "cosit") == 15011) . orderBy( "hour" )
jun15 . groupBy( ) . agg(F.sum( " speed" ) . alias( "count " ) ) \

            . withC01umnRenamed( ' count ' ,          ' totalSpeed' ) \

            . withC01umn( ' Average Speed' ,           (F. col( ' totalSpeed' ) / totalSpeed) )

. show( )

print( "Avg Speed between Junction 16 and 17 " jun16 = M50DF.  "cosit") 15012) . orderBy( "hour" )
jun16 . groupBy( ) . agg(F.sum( " speed" ) . alias( "count " ) ) \

            . withC01umnRenamed ( count '' total Speed' ) \

. withC01umn( ' Average Speed' ,           (F. col( ' total Speed' ) / total Speed) ) . show( )
```

```
Sum of Speeds between junction 3 and junction 17

.0

Avg Speed between Junction 3 and 4


I totalSpeed I     Average Speed I


| 1168870 .0 1 0.06380070482238506 1


Avg Speed between Junction 4 and 5


I totalSpeed I     Average Speed I
```

| 3900959 .0 1 0. 21292695824448094 1

QUESTION 6

ß Question 6 —Calculate the top 10 locations with highest number of counts of HGVs (class) .

HGV_ART DF = "classname") —=- 'HGV_ART')

HGV RIG DF = M50DF. filter (col("classname")   'HGV RIG')

HGV DF -- HGV ART_DF. join(HGV RIG DF, [' cosit'

HighestHGV = HGV DF.   ' cosit '

. count( ) \

. withC01 umnRenamed (' count '   ' Total') \

. orderBy (COI (' cos it ') .desc( ))

# print the Top 10 locations with highest number of counts of HGVs

HighestHGV. orderBy (col (' Total ') .desc( )) .show(10)

Map the COSITs with their names given on the map

—> Ballymun, Ballymun

—> Ballymun, finglas

—> M50 Between Jn06 N03/M50 and Jn05 N02/M50, Finglas, Co. Dublin

—> M50 Between Jn07 N04/M50 and Jn09 N07/M50 Red Cow, Palmerstown, Co. Dublin

—> M50 Between Jn10 — Ballymount and Jnll — Tymon, Co. Dublin

—> M50 Between Jnll Tallaght and Jn12 Firhouse, Co. Dublin

—> M50 Between Jn12 Firhouse and Jn13 Dundrum, Balinteer, Co. Dublin

—> M50 Between Jn06 N03/M50 and Jn07 N04/M50, Castleknock, Co. Dublin

—> M50 Between Jn09 N07/M50 Red Cow and Jn10 Ballymount, Ballymount, Co. Dublin

15010 —> M50 Between Jn14 Dun Laoghaire and Jn15 Carrickmines, Cabinteely, Co. Dublin

In [246] :

I cositl Total I

15081 1747351

1 1518401

1 141361 1

1501 1 1096201

15001 510861