```
+-----------------+-------------------+------------------------------+
|model            |x1                 |x2                            |
+-----------------+-------------------+------------------------------+
|Mazda RX4        |{21.0, 6, 160.0, 110}|{3.9, 2.62, 16.46, 0, 1, 4, 4}  |
|Mazda RX4 Wag    |{21.0, 6, 160.0, 110}|{3.9, 2.875, 17.02, 0, 1, 4, 4} |
|Datsun 710       |{22.8, 4, 108.0, 93} |{3.85, 2.32, 18.61, 1, 1, 4, 1} |
|Hornet 4 Drive   |{21.4, 6, 258.0, 110}|{3.08, 3.215, 19.44, 1, 0, 3, 1}|
|Hornet Sportabout|{18.7, 8, 360.0, 175}|{3.15, 3.44, 17.02, 0, 0, 3, 2} |
+-----------------+-------------------+------------------------------+
only showing top 5 rows
```

5. TODO Exercises

TODO 1: In the previous tutorial, we extracted continuous patterns of size k from a DNA string. Now, suppose we have a DNA sequence that appears more than 10 times, as provided in the cell below. Please convert this data into a DataFrame with two columns: (sequence, count).

```
# TODO 1: Transform the provided data into a DataFrame with two columns: (sequence, count).

from operator import add
from pyspark.sql import Row

data = [
    ("ctaag", 103),
    ("gccta", 96),
    ("cctaa", 95),
    ("aagcc", 94),
    ("agcct", 94),
    ("taagc", 93),
    ("ttttt", 30),
    ("aaaaa", 20),
    ("aaaat", 20),
    ("atttt", 18),
    ("aaatt", 16),
    ("actaa", 15),
    ("ttttc", 14),
    ("taaga", 13),
    ("aattt", 13),
    ("tttta", 12),
    ("gtttt", 11)
]

rdd = ss.sparkContext.parallelize(data)

df = rdd.map(lambda x: Row(sequence=x[0], count=x[1])).toDF()
df.show()
```

```
+--------+-----+
|sequence|count|
+--------+-----+
|   ctaag|  103|
|   gccta|   96|
|   cctaa|   95|
|   aagcc|   94|
|   agcct|   94|
|   taagc|   93|
|   ttttt|   30|
|   aaaaa|   20|
|   aaaat|   20|
|   atttt|   18|
|   aaatt|   16|
|   actaa|   15|
|   ttttc|   14|
|   taaga|   13|
|   aattt|   13|
|   tttta|   12|
|   gtttt|   11|
+--------+-----+
```

TODO 2: Given two datasets hills2000.txt and hills.txt, they are available in blackboard, please download them and upload to Colab or placing in the approporiate directory for loading.

Calculate the average speed (in miles per hour) for each race in both datasets, combining them into a single DataFrame, and identify the top 5 races with the highest average speed.

For races in hills2000.txt that have both time (male) and timef (female) columns, use the average of the two times to compute speed.

Please note that in this question, we may need to use the `col` SQL function.

You can import it using `from pyspark.sql.functions import col`

Reference: https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.functions.col.html

TODO 2.1: Load the dataset from `hills2000.txt` and create the RDDs for races that contain both the male time and female time columns. Then, compute the average time using the formula `(time+timef)/2` and store the result in a new column called avg_speed.

```
from pyspark.sql.functions import col
hills2000_df = ss.read.csv("hills2000.txt", header=True, inferSchema=True)
hills2000_df = hills2000_df.withColumn('avg_time', (col('time')+col('timef'))/2)
hills2000_df.show()
```

```
+--------------------+----+-----+-----------------+-----------------+-------------------+
|            rownames|dist|climb|             time|            timef|           avg_time|
+--------------------+----+-----+-----------------+-----------------+-------------------+
|       Tiso Carnethy| 6.0| 2500|0.782222222222222|0.919166666666667| 0.8506944444444445|
|             Criffel| 7.0| 1800|0.793333333333333| 1.00333333333333| 0.8983333333333315|
|          Chapelgill| 1.5| 1400|0.314444444444444|0.376666666666667| 0.3455555555555555|
|        Norman's Law| 5.0|  700|0.464166666666667|0.609166666666667|  0.536666666666667|
|        Craig Dunain| 6.0|  900|0.546111111111111|0.625833333333333|  0.585972222222222|
|         Knockfarrel| 5.0| 1200|0.623333333333333|0.734166666666667|            0.67875|
|              Screel| 4.0| 1300|0.458888888888889|0.543611111111111|            0.50125|
|   O.P.S. Clachnaben|10.5| 3500| 1.27888888888889| 1.48777777777778|  1.383333333333335|
|       Whangie Whizz| 3.5|  800|0.395555555555556|0.464166666666667|0.42986111111111147|
|       Stuc A'Chroin|14.0| 5000| 1.98333333333333| 2.39888888888889|   2.19111111111111|
|         29th Dumyat| 5.0| 1250|0.552777777777778|0.637777777777778|  0.595277777777778|
|          Ben Lomond| 9.0| 3192| 1.03777777777778| 1.19916666666667|  1.118472222222225|
|            Kinnoull| 4.0|  800|0.387222222222222|           0.4525|  0.419861111111111|
|            Goatfell| 8.0| 2866| 1.22777777777778| 1.52638888888889|  1.377083333333335|
|Scottish Nuclear ...| 3.5| 1250|0.431111111111111|             0.54| 0.4855555555555555|
|          Ardoch Rig| 7.0|  600|0.678888888888889|0.837222222222222| 0.7580555555555555|
|              Scolty| 5.0|  800|0.495833333333333|0.579166666666667| 0.5375000000000001|
|           Cornalees| 5.5|  800|0.618333333333333|             NULL|               NULL|
|        12 Trig Trog|46.0| 7500| 8.30694444444445| 13.5477777777778| 10.927361111111125|
|         Kilpatricks| 6.0| 1400|0.739722222222222|0.874166666666667| 0.8069444444444445|
+--------------------+----+-----+-----------------+-----------------+-------------------+
only showing top 20 rows
```

TODO 2.2: Once we have calculated the average time from Task 2.1, we can determine the average speed by dividing distance by the `avg_time`. Store the resulting value in a new column named `avg_speed`.

```
hills2000_df = hills2000_df.withColumn('speed', col('dist')/col('avg_time'))
hills2000_df.show()
```

```
+--------------------+----+-----+-----------------+-----------------+-------------------+------------------+
|            rownames|dist|climb|             time|            timef|           avg_time|             speed|
+--------------------+----+-----+-----------------+-----------------+-------------------+------------------+
|       Tiso Carnethy| 6.0| 2500|0.782222222222222|0.919166666666667| 0.8506944444444445| 7.053061224489795|
|             Criffel| 7.0| 1800|0.793333333333333| 1.00333333333333| 0.8983333333333315| 7.792207792207808|
|          Chapelgill| 1.5| 1400|0.314444444444444|0.376666666666667| 0.3455555555555555| 4.340836012861737|
|        Norman's Law| 5.0|  700|0.464166666666667|0.609166666666667|  0.536666666666667| 9.316770186335399|
|        Craig Dunain| 6.0|  900|0.546111111111111|0.625833333333333|  0.585972222222222|10.239393221142455|
|         Knockfarrel| 5.0| 1200|0.623333333333333|0.734166666666667|            0.67875| 7.366482504604052|
|              Screel| 4.0| 1300|0.458888888888889|0.543611111111111|            0.50125| 7.980049875311721|
|   O.P.S. Clachnaben|10.5| 3500| 1.27888888888889| 1.48777777777778|  1.383333333333335| 7.590361445783123|
|       Whangie Whizz| 3.5|  800|0.395555555555556|0.464166666666667|0.42986111111111147| 8.142164781906294|
|       Stuc A'Chroin|14.0| 5000| 1.98333333333333| 2.39888888888889|   2.19111111111111| 6.389452332657204|
|         29th Dumyat| 5.0| 1250|0.552777777777778|0.637777777777778|  0.595277777777778|  8.39944003733084|
|          Ben Lomond| 9.0| 3192| 1.03777777777778| 1.19916666666667|  1.118472222222225| 8.046690674282857|
|            Kinnoull| 4.0|  800|0.387222222222222|           0.4525|  0.419861111111111| 9.526959973536224|
|            Goatfell| 8.0| 2866| 1.22777777777778| 1.52638888888889|  1.377083333333335| 5.809379727685318|
|Scottish Nuclear ...| 3.5| 1250|0.431111111111111|             0.54| 0.4855555555555555| 7.208237986270023|
|          Ardoch Rig| 7.0|  600|0.678888888888889|0.837222222222222| 0.7580555555555555| 9.234151703920851|
|              Scolty| 5.0|  800|0.495833333333333|0.579166666666667| 0.5375000000000001| 9.302325581395348|
|           Cornalees| 5.5|  800|0.618333333333333|             NULL|               NULL|              NULL|
|        12 Trig Trog|46.0| 7500| 8.30694444444445| 13.5477777777778| 10.927361111111125| 4.209616533421452|
|         Kilpatricks| 6.0| 1400|0.739722222222222|0.874166666666667| 0.8069444444444445| 7.435456110154905|
+--------------------+----+-----+-----------------+-----------------+-------------------+------------------+
only showing top 20 rows
```

TODO 2.3: Perform the same calculation on another dataset, `hills.txt`, by dividing the distance by the time to obtain the speed.

```
hills_df = ss.read.csv("hills.txt", header=True, inferSchema=True)
hills_df = hills_df.withColumn('speed', col('dist')/col('time'))
hills_df.show()
```

```
+------------+----+-----+-----------------+------------------+
|    rownames|dist|climb|             time|             speed|
+------------+----+-----+-----------------+------------------+
```

```
| Greenmantle| 2.4|  650|0.268055555555556| 8.953367875647654|
|    Carnethy| 6.0| 2500|0.805833333333333| 7.445708376421927|
|Craig Dunain| 6.0|  900|0.560833333333333|10.698365527488862|
|     Ben Rha| 7.5|  800|             0.76| 9.868421052631579|
|  Ben Lomond| 8.0| 3070| 1.03777777777778|7.7087794432548025|
|    Goatfell| 8.0| 2866| 1.22027777777778| 6.555884361484168|
|Bens of Jura|16.0| 7500| 3.41027777777778| 4.691699926692185|
| Cairnpapple| 6.0|  800|0.606111111111111| 9.899175068744272|
|      Scolty| 5.0|  800|0.495833333333333|10.084033613445385|
|    Traprain| 6.0|  650|           0.6625| 9.056603773584905|
| Lairig Ghru|28.0| 2100| 3.21111111111111| 8.719723183391007|
|      Dollar| 5.0| 2000|           0.7175| 6.968641114982578|
|     Lomonds| 9.5| 2200| 1.08333333333333| 8.769230769230797|
| Cairn Table| 6.0|  500|0.735555555555556| 8.157099697885192|
|  Eildon Two| 4.5| 1500|0.448888888888889|10.024752475247523|
|   Cairngorm|10.0| 3000| 1.20416666666667| 8.304498269896172|
| Seven Hills|14.0| 2200| 1.64027777777778| 8.535139712108371|
|  Knock Hill| 3.0|  350| 1.31083333333333| 2.288620470438658|
|  Black Hill| 4.5| 1000|0.290277777777778|15.502392344497595|
|  Creag Beag| 5.5|  600|0.542777777777778|10.133060388945747|
+------------+----+-----+-----------------+------------------+
only showing top 20 rows
```

TODO 2.4: Merge both dataframes into one, ensuring it contains the columns (`rownames`, `climb`, `dist`, and `avg_speed`).

```
hills_speed = hills_df.select(col('rownames'), col('climb'), col('dist'), col('speed'))
hills_2000_speed = hills2000_df.select(col('rownames'), col('climb'), col('dist'), col('speed'))

hills_union = hills_speed.union(hills_2000_speed)
hills_union.show()
```

```
+------------+-----+----+------------------+
|    rownames|climb|dist|             speed|
+------------+-----+----+------------------+
| Greenmantle|  650| 2.4| 8.953367875647654|
|    Carnethy| 2500| 6.0| 7.445708376421927|
|Craig Dunain|  900| 6.0|10.698365527488862|
|     Ben Rha|  800| 7.5| 9.868421052631579|
|  Ben Lomond| 3070| 8.0|7.7087794432548025|
|    Goatfell| 2866| 8.0| 6.555884361484168|
|Bens of Jura| 7500|16.0| 4.691699926692185|
| Cairnpapple|  800| 6.0| 9.899175068744272|
|      Scolty|  800| 5.0|10.084033613445385|
|    Traprain|  650| 6.0| 9.056603773584905|
| Lairig Ghru| 2100|28.0| 8.719723183391007|
|      Dollar| 2000| 5.0| 6.968641114982578|
|     Lomonds| 2200| 9.5| 8.769230769230797|
| Cairn Table|  500| 6.0| 8.157099697885192|
|  Eildon Two| 1500| 4.5|10.024752475247523|
|   Cairngorm| 3000|10.0| 8.304498269896172|
| Seven Hills| 2200|14.0| 8.535139712108371|
|  Knock Hill|  350| 3.0| 2.288620470438658|
|  Black Hill| 1000| 4.5|15.502392344497595|
|  Creag Beag|  600| 5.5|10.133060388945747|
+------------+-----+----+------------------+
only showing top 20 rows
```

TODO 2.5: Retrieve the top 5 races with the highest average speed, sorted in descending order. If there are multiple entries for the same rowname, we should calculate the average speed.

```
hills_union.groupby("rownames").avg('speed').sort('avg(speed)', ascending=False).show(5)
```

```
+------------+------------------+
|    rownames|        avg(speed)|
+------------+------------------+
|      Acmony|14.319809069212397|
|  Black Hill|11.832828825310022|
|Kildcon Hill|11.285266457680265|
|  Caerketton|10.982306284319701|
|   Largo Law|10.501750291715288|
+------------+------------------+
only showing top 5 rows
```

TODO 2.6: Next, we will calculate the difficulty score for each race by dividing the `climb` by the `distance`. The results will be stored in a new column named `diff_score`.

```
hills_union_diff = hills_union.withColumn('diff_score', col('climb')/col('dist'))
hills_union_diff.show()
```

```
+------------+-----+----+------------------+------------------+
|    rownames|climb|dist|             speed|         diff_score|
+------------+-----+----+------------------+------------------+
| Greenmantle|  650| 2.4| 8.953367875647654|270.83333333333337|
|    Carnethy| 2500| 6.0| 7.445708376421927| 416.6666666666667|
|Craig Dunain|  900| 6.0|10.698365527488862|             150.0|
|     Ben Rha|  800| 7.5| 9.868421052631579|106.66666666666667|
|  Ben Lomond| 3070| 8.0|7.7087794432548025|             383.75|
|    Goatfell| 2866| 8.0| 6.555884361484168|             358.25|
|Bens of Jura| 7500|16.0| 4.691699926692185|             468.75|
| Cairnpapple|  800| 6.0| 9.899175068744272|133.33333333333334|
|      Scolty|  800| 5.0|10.084033613445385|             160.0|
|    Traprain|  650| 6.0| 9.056603773584905|108.33333333333333|
| Lairig Ghru| 2100|28.0| 8.719723183391007|              75.0|
|      Dollar| 2000| 5.0| 6.968641114982578|             400.0|
|     Lomonds| 2200| 9.5| 8.769230769230797|231.57894736842104|
| Cairn Table|  500| 6.0| 8.157099697885192| 83.33333333333333|
|  Eildon Two| 1500| 4.5|10.024752475247523| 333.3333333333333|
|   Cairngorm| 3000|10.0| 8.304498269896172|             300.0|
| Seven Hills| 2200|14.0| 8.535139712108371|157.14285714285714|
|  Knock Hill|  350| 3.0| 2.288620470438658|116.66666666666667|
|  Black Hill| 1000| 4.5|15.502392344497595|222.22222222222223|
|  Creag Beag|  600| 5.5|10.133060388945747| 109.0909090909091|
+------------+-----+----+------------------+------------------+
only showing top 20 rows
```

TODO 2.7: Next, we need to categorize the difficulty levels as 'Easy', 'Moderate', or 'Difficult'. If the diff_score is less than 200, it will be classified as 'Easy'; if it's less than 350, it will be classified as 'Moderate'; otherwise, it will be classified as 'Difficult'. For this task, you may need to use the when function.

The when() function takes a condition and a value: if the condition evaluates to True, the specified value is returned. You can chain multiple when() calls to handle several conditions sequentially.

Ref: https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.functions.when.html

```
from pyspark.sql.functions import when
hills_union_diff_level = hills_union_diff.withColumn('level', when(col('diff_score') < 200, 'Easy').when(col('di
hills_union_diff_level.show()
```

```
+------------+-----+----+------------------+------------------+---------+
|    rownames|climb|dist|             speed|         diff_score|    level|
+------------+-----+----+------------------+------------------+---------+
| Greenmantle|  650| 2.4| 8.953367875647654|270.83333333333337| Moderate|
|    Carnethy| 2500| 6.0| 7.445708376421927| 416.6666666666667|Difficult|
|Craig Dunain|  900| 6.0|10.698365527488862|             150.0|     Easy|
|     Ben Rha|  800| 7.5| 9.868421052631579|106.66666666666667|     Easy|
|  Ben Lomond| 3070| 8.0|7.7087794432548025|             383.75|Difficult|
|    Goatfell| 2866| 8.0| 6.555884361484168|             358.25|Difficult|
|Bens of Jura| 7500|16.0| 4.691699926692185|             468.75|Difficult|
| Cairnpapple|  800| 6.0| 9.899175068744272|133.33333333333334|     Easy|
|      Scolty|  800| 5.0|10.084033613445385|             160.0|     Easy|
|    Traprain|  650| 6.0| 9.056603773584905|108.33333333333333|     Easy|
| Lairig Ghru| 2100|28.0| 8.719723183391007|              75.0|     Easy|
|      Dollar| 2000| 5.0| 6.968641114982578|             400.0|Difficult|
|     Lomonds| 2200| 9.5| 8.769230769230797|231.57894736842104| Moderate|
| Cairn Table|  500| 6.0| 8.157099697885192| 83.33333333333333|     Easy|
|  Eildon Two| 1500| 4.5|10.024752475247523| 333.3333333333333| Moderate|
|   Cairngorm| 3000|10.0| 8.304498269896172|             300.0| Moderate|
| Seven Hills| 2200|14.0| 8.535139712108371|157.14285714285714|     Easy|
|  Knock Hill|  350| 3.0| 2.288620470438658|116.66666666666667|     Easy|
|  Black Hill| 1000| 4.5|15.502392344497595|222.22222222222223| Moderate|
|  Creag Beag|  600| 5.5|10.133060388945747| 109.0909090909091|     Easy|
+------------+-----+----+------------------+------------------+---------+
only showing top 20 rows
```

TODO 2.8: Finally, displaying the count of races in each difficulty category.

```
hills_union_diff_level.groupby("level").count().show()
```

```
+---------+-----+
|    level|count|
+---------+-----+
|Difficult|   29|
|     Easy|   29|
| Moderate|   33|
+---------+-----+
```

TODO 3: Write a module to read setenceData as RDD, then split the text by space and lowercase the word. Finally, do the following counts:

- **No. of Document**: Number of document
- **Term_Frequency**: Words count of each document in any output format. E.g., ((Doc ID, word), count)((0, 'studying'), 2), (1, 'studying;), 1)
- **Document_Frequency**: No. of document containing a word. E.g., (Word, Count, ('studying', 2)

Hints: You may need to use the following functions (also look into Example 3.4.5):

- `lower()`
- `split()`
- `flatMapValues()`
- `distinct()`
- `count()`
- `countByKey()`
- `countByValue()`

```
#TODO 3. Write a module to read setenceData as RDD, then split the text by space and lowercase the word.
#        Finally, do the following counts:
#        No. of Document: Number of document
#        Term_Frequency: Words count of each document in any output format. E.g., ((Doc ID, word), count)((0, 'st
#.       Document_Frequency: No. of document containing a word. E.g., (Word, Count, ('studying', 2)
sentenceData = ss.createDataFrame([
    (0, "I am Billy studying in Computer and I am also studying in Business Course"),
    (1, "Ada is studying in Computer"),
    (2, "Billy and Ada know each other in a computer course course")],
    ["document", "sentence"])

def text_processing(text):
  return [w.lower() for w in text.split()]

## TODO 3.1 print out the number of documents.
tokenized_text = sentenceData.rdd.map(lambda x: (x.document, text_processing(x.sentence)))
num_of_documents = len(tokenized_text.collect())
print('Number of documents:', num_of_documents)

## TODO 3.2 count words in each document
wordItems = tokenized_text.flatMapValues(lambda x: x).countByValue().items()

print("\nTerm_Frequency:")
for word, cnt in wordItems:
  print(word[1]+":", cnt, "in D" + str(word[0]))

print("\nDocument Frequency:")
## TODO 3.3 No of documents containing a term
df = tokenized_text.flatMapValues(lambda x: x).distinct().map(lambda x:(x[1], x[0])).countByKey()
df.items()
for word, cnt in df.items():
  print(word, ":", cnt, sep='', end='|')
```

```
Number of documents: 3
dict_items([('i', 1), ('am', 1), ('billy', 2), ('studying', 2), ('in', 3), ('computer', 3), ('and', 2), ('also',
1), ('business', 1), ('course', 2), ('ada', 2), ('is', 1), ('know', 1), ('each', 1), ('other', 1), ('a', 1)])
```