

Module 8

PySpark + ELK + Kafka

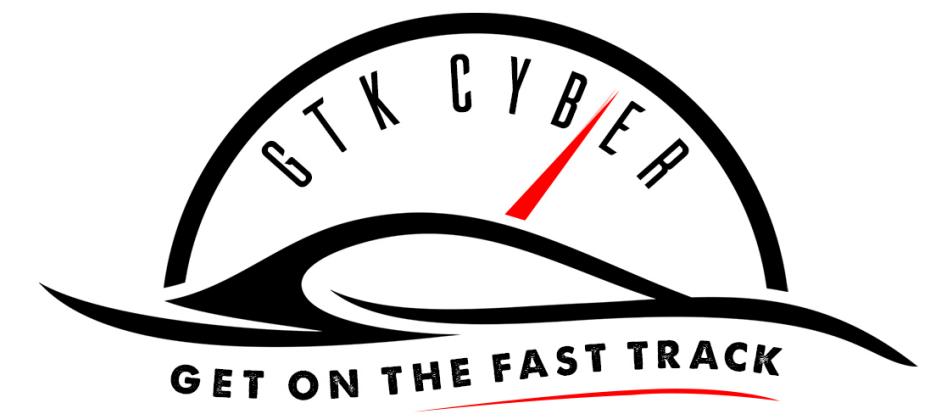
Distributed Computing and Streaming Analytics



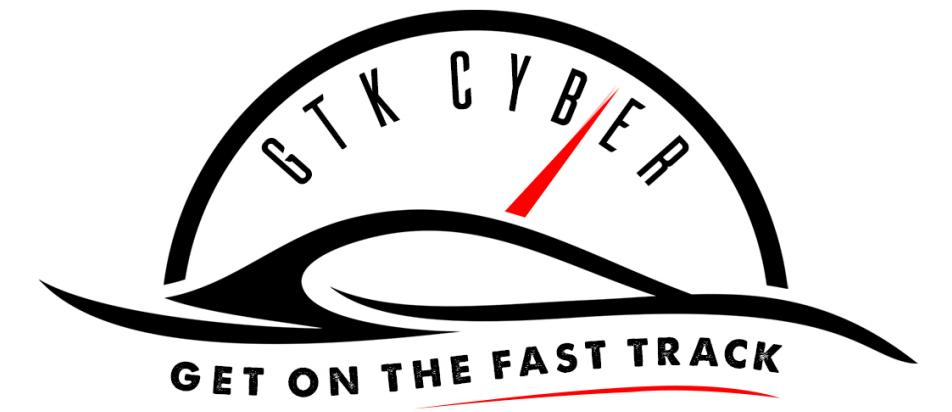
LET'S SOLVE THIS PROBLEM BY
USING THE BIG DATA NONE
OF US HAVE THE SLIGHTEST
IDEA WHAT TO DO WITH



© marketoonist.com



Data

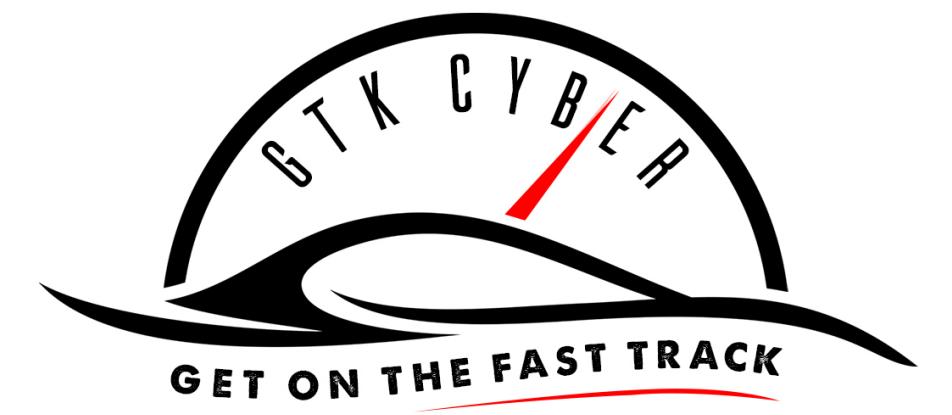


Data

I LOVE IT WHEN



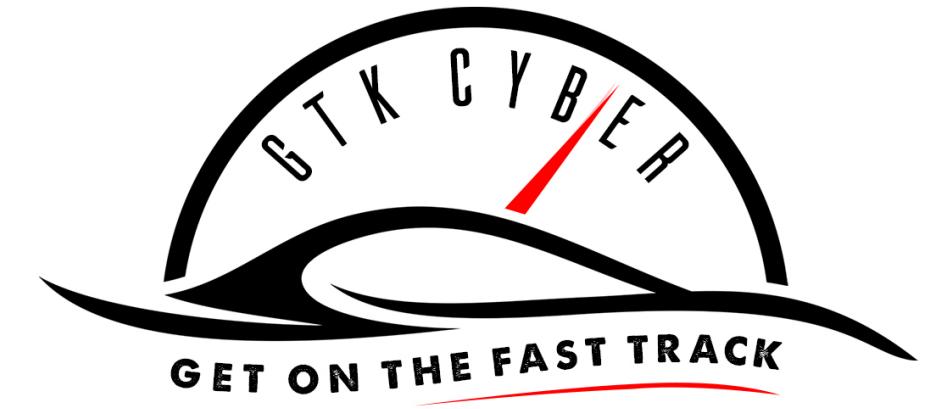
YOU CALL ME BIG
DATA



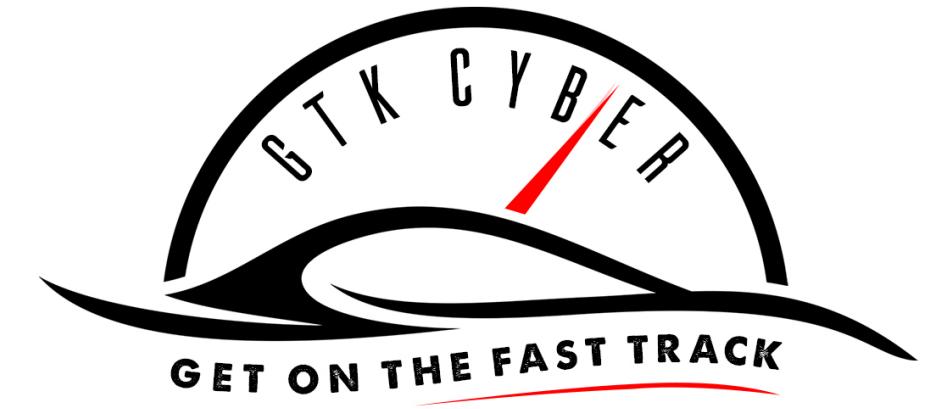
What is big data?



Big data generally refers to data sets which are too large to process using conventional applications.

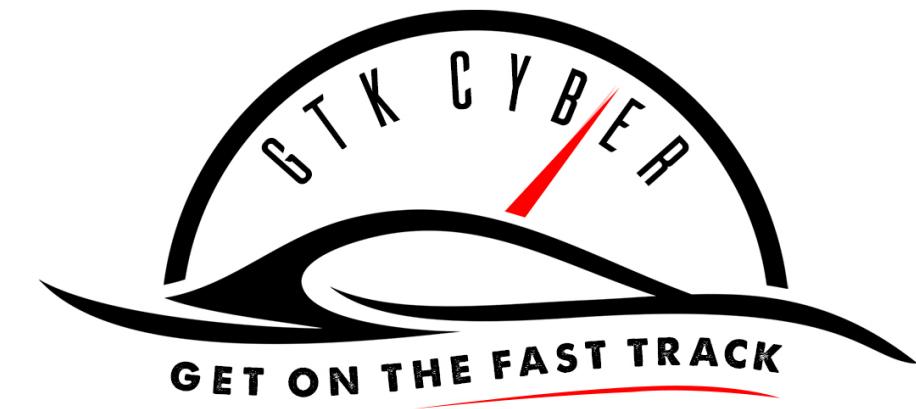


Volume: The actual
amount of data



Volume: The actual amount of data

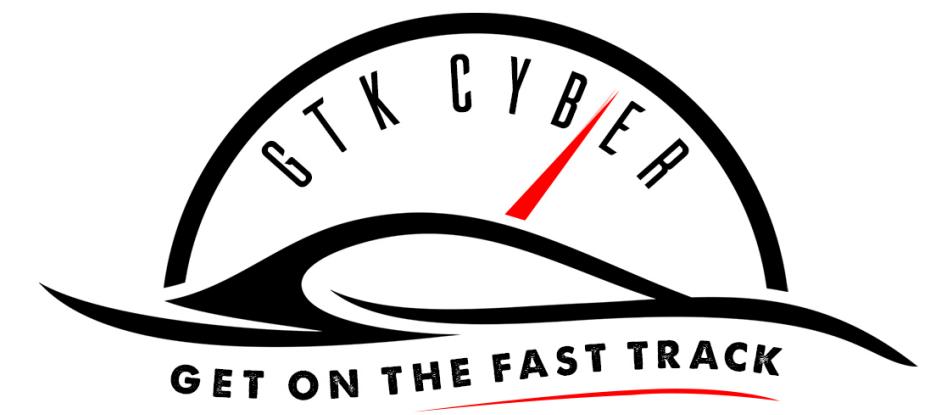
Velocity: The speed at which the data is generated



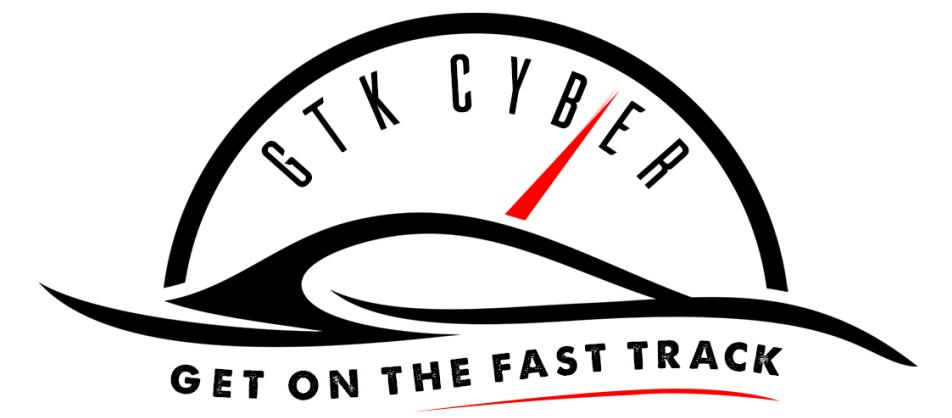
Volume: The actual amount of data

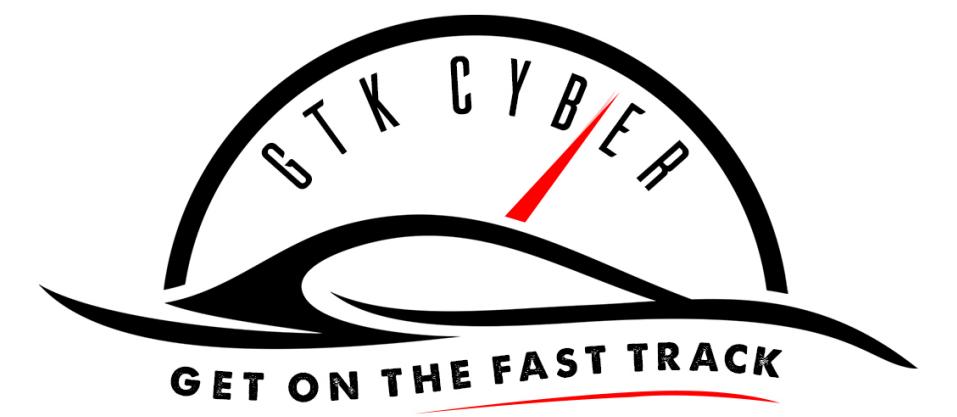
Velocity: The speed at which the data is generated

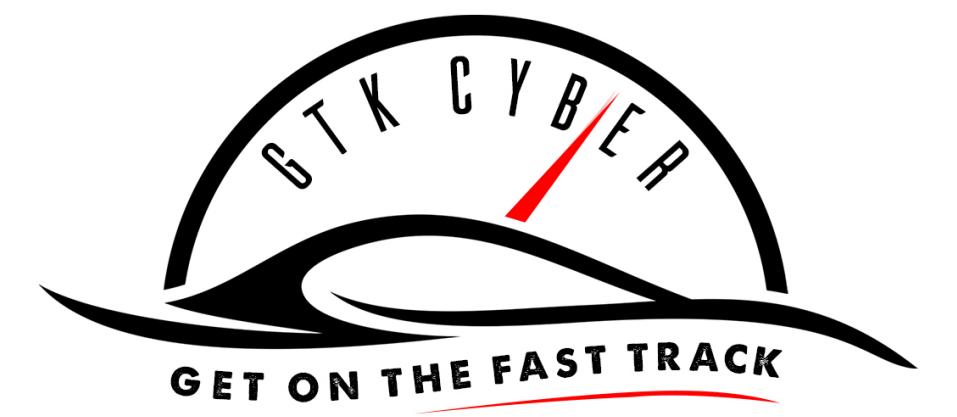
Variety: Data arrives in many different forms



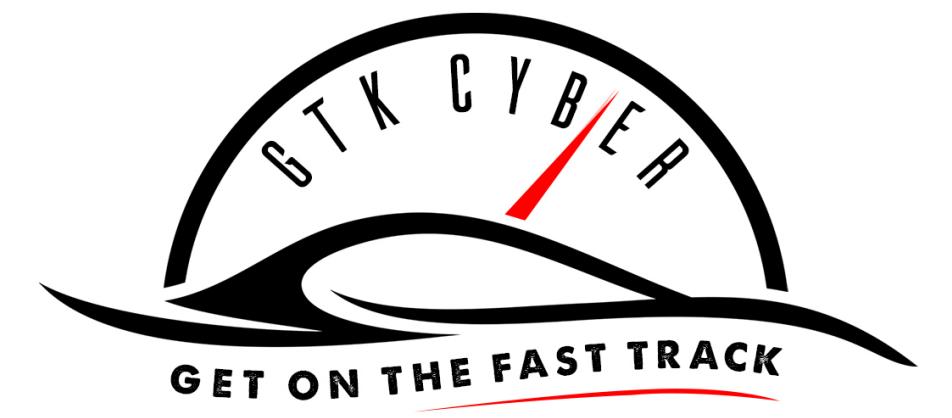
<http://internetlivestats.com/one-second/>

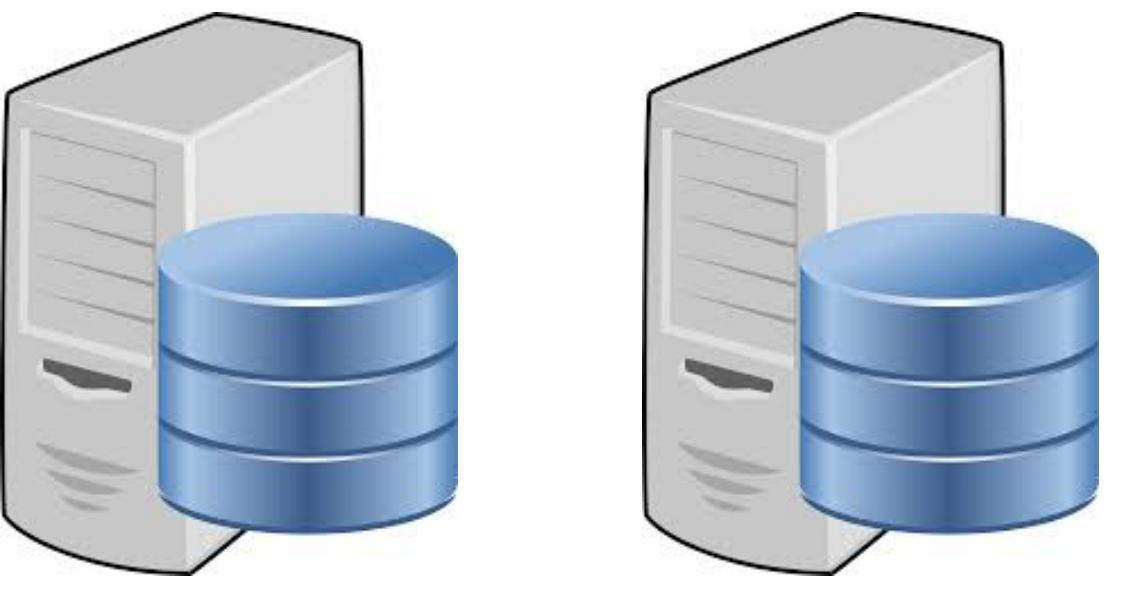
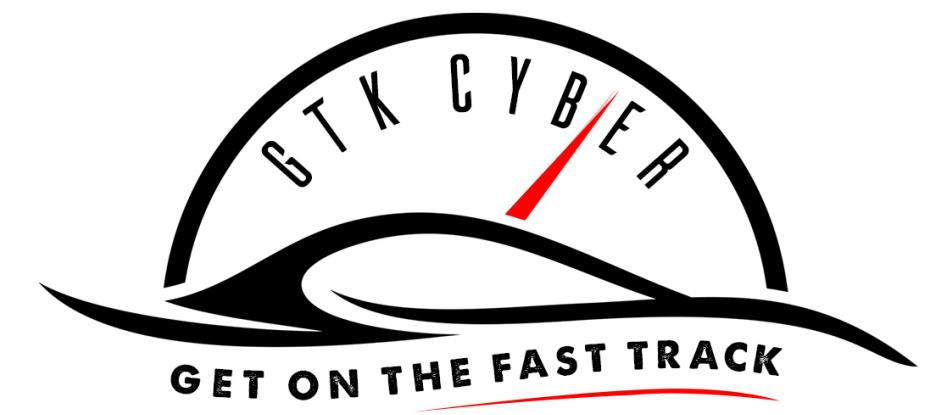


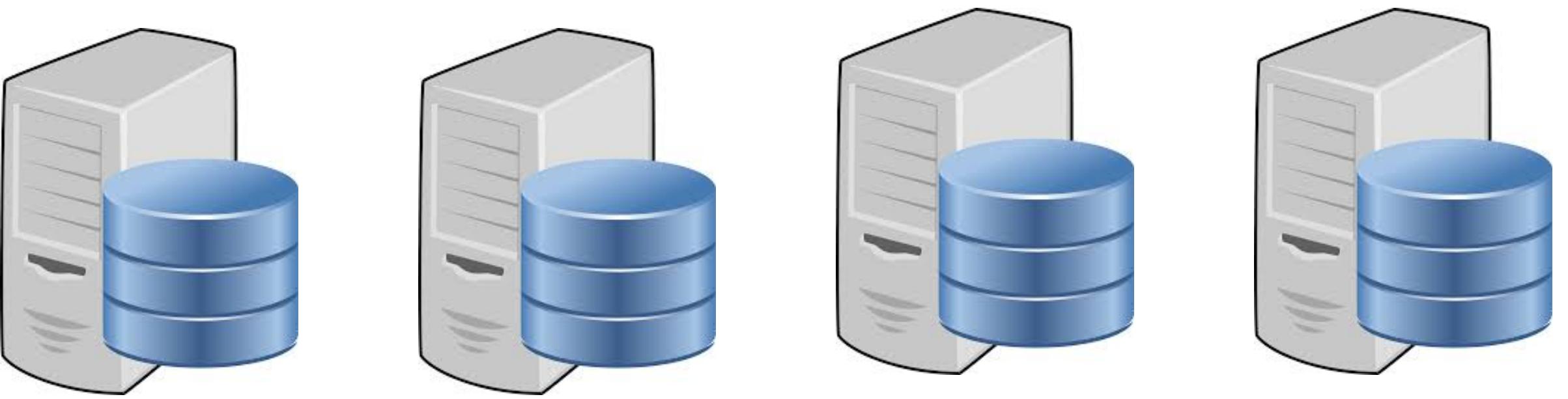
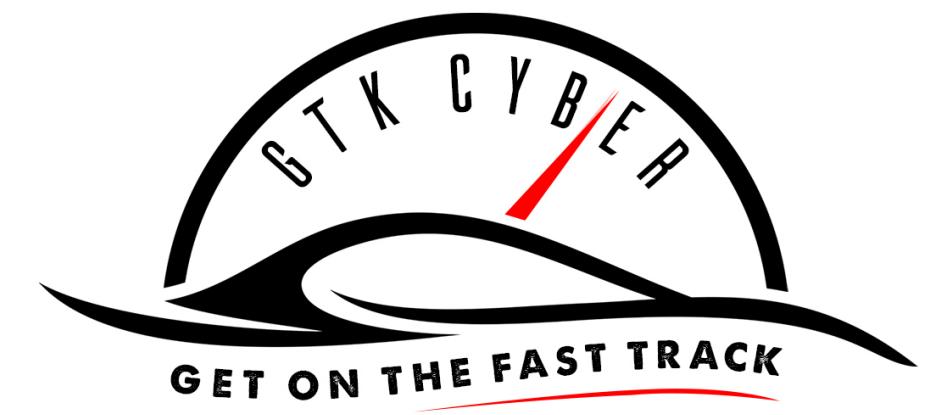


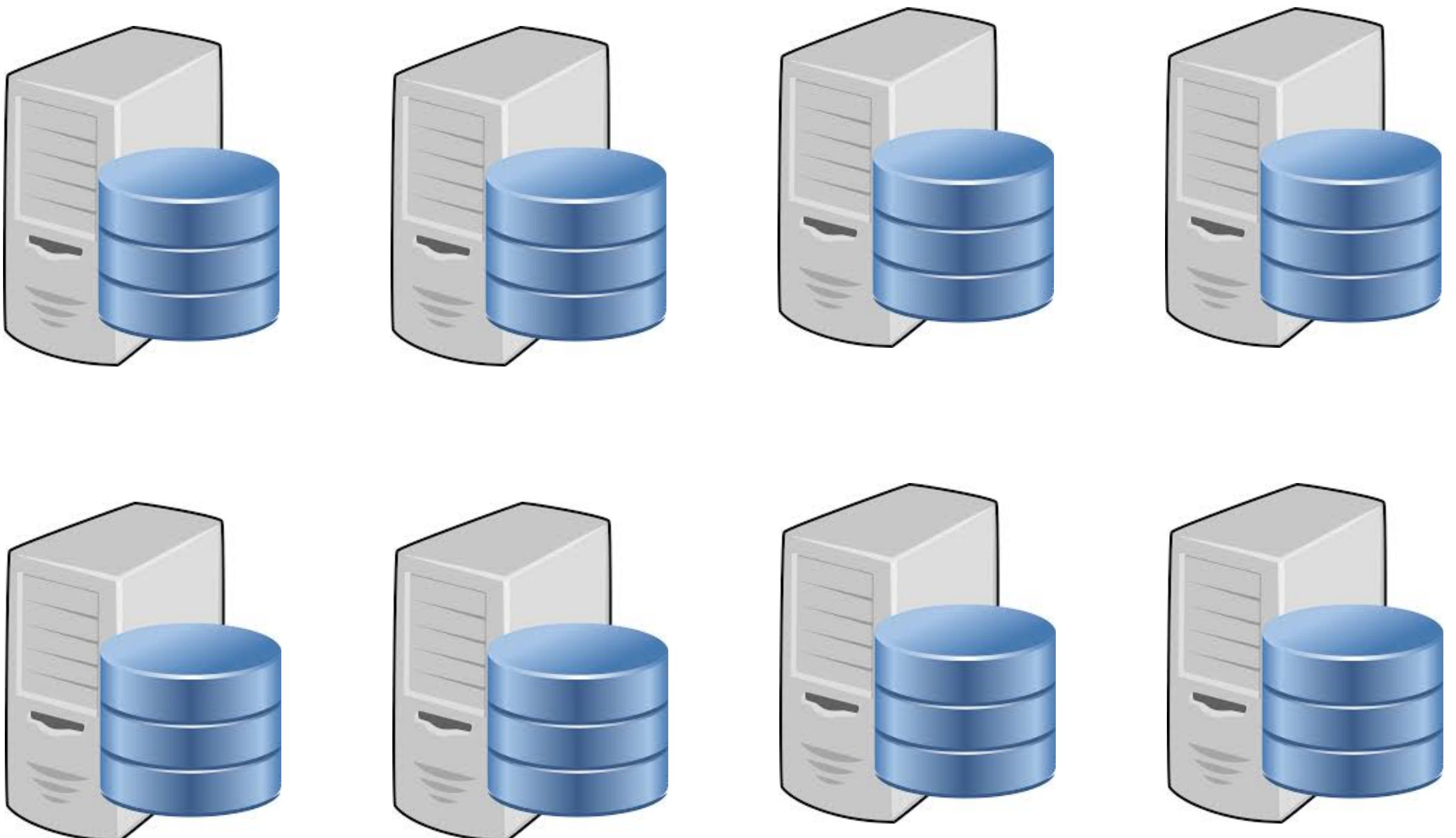
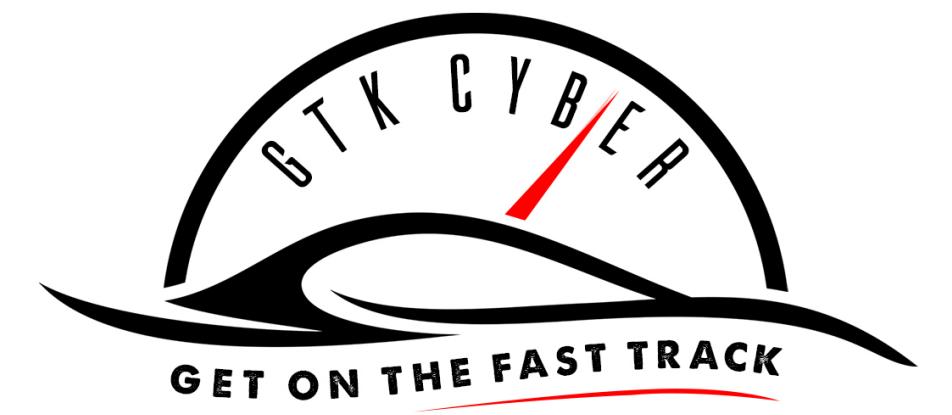






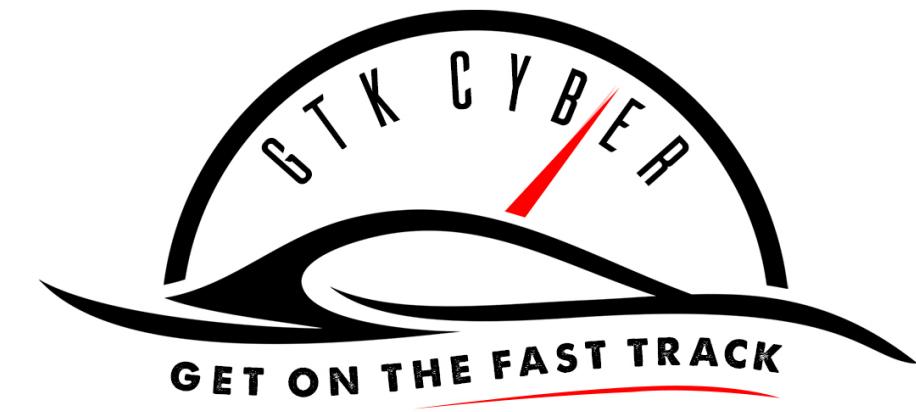






Cloud Computing





MapReduce

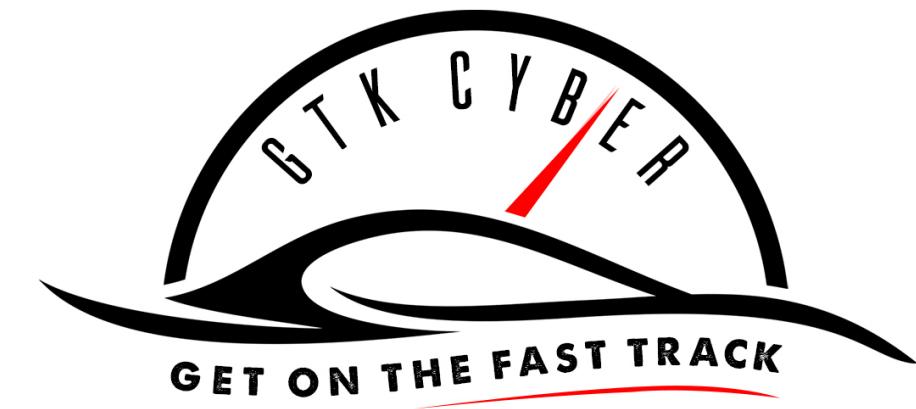
<http://research.google.com/archive/mapreduce.html>



MapReduce in Python

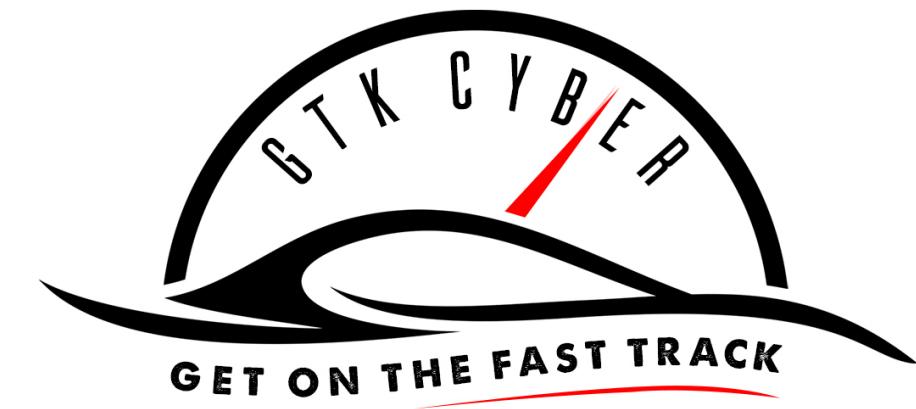
```
import sys

for line in sys.stdin:
    line = line.strip()
    keys = line.split()
    for key in keys:
        value = 1
        print( "%s\t%d" % (key, value) )
```



MapReduce in Python

```
away--you 1
may 1
do1
practically 1
ANYTHING 1
with1
public 1
domain 1
eBooks. 1
Redistribution 1
is1
subject 1
```



MapReduce in Python

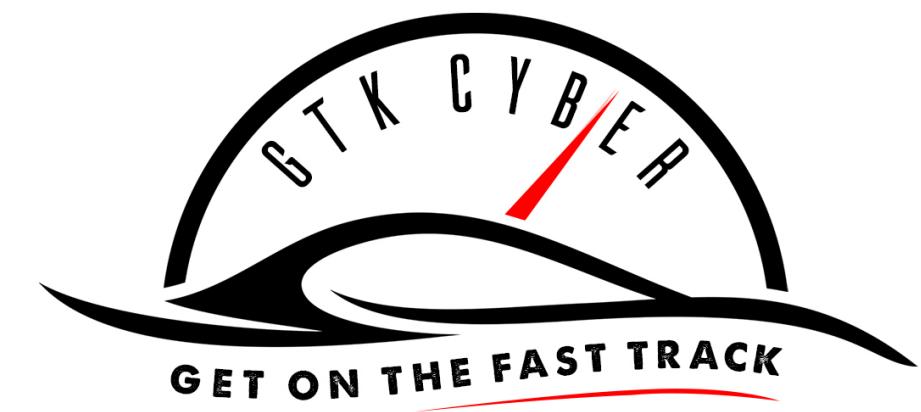
```
import sys

last_key = None
running_total = 0

for input_line in sys.stdin:
    input_line = input_line.strip()
    this_key, value = input_line.split("\t", 1)
    value = int(value)

    if last_key == this_key:
        running_total += value
    else:
        if last_key:
            print( "%s\t%d" % (last_key, running_total) )
        running_total = value
        last_key = this_key

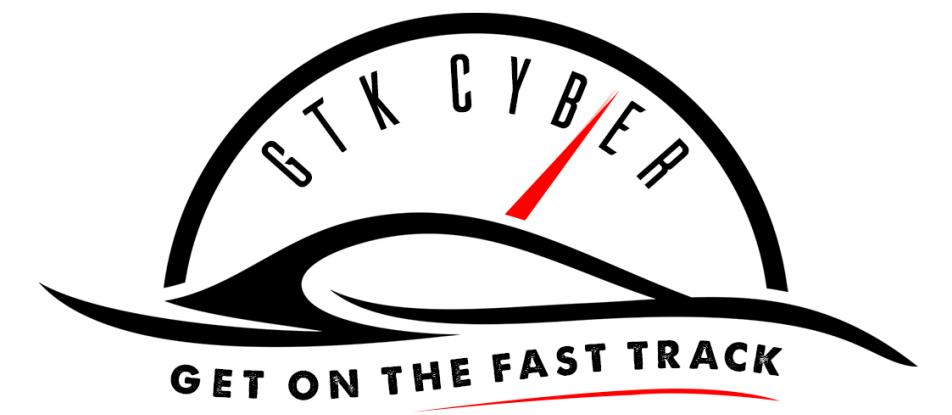
    if last_key == this_key:
        print( "%s\t%d" % (last_key, running_total) )
```



MapReduce in Python

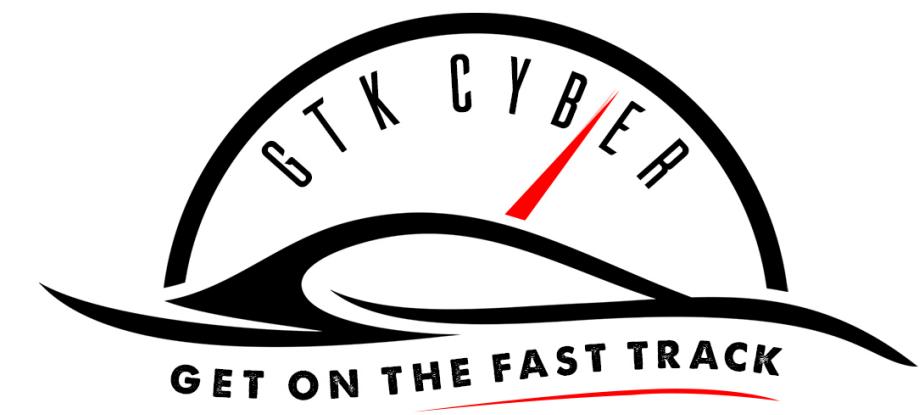
```
cat sherlock.txt | ./map.py | sort | ./reduce.py
```

```
available 1
avenue 1
avenue. 1
average 3
averse 5
aversion 2
avert 3
averted 1
avoid 3
avoided 2
...
```

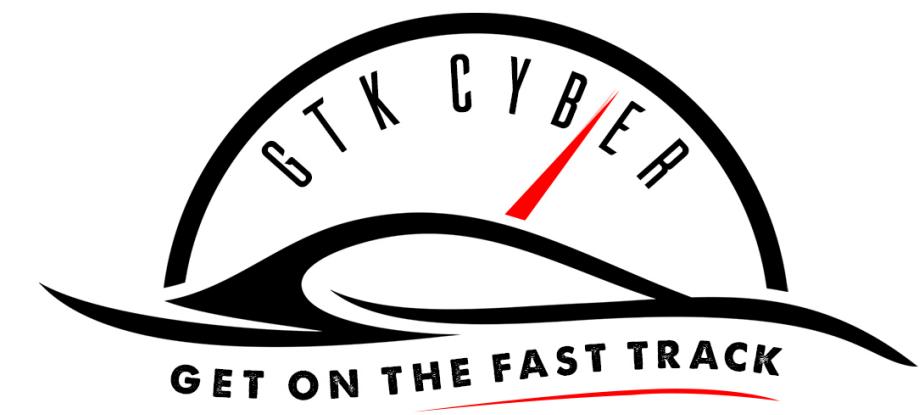


Map Reduce is Hard...Really Hard





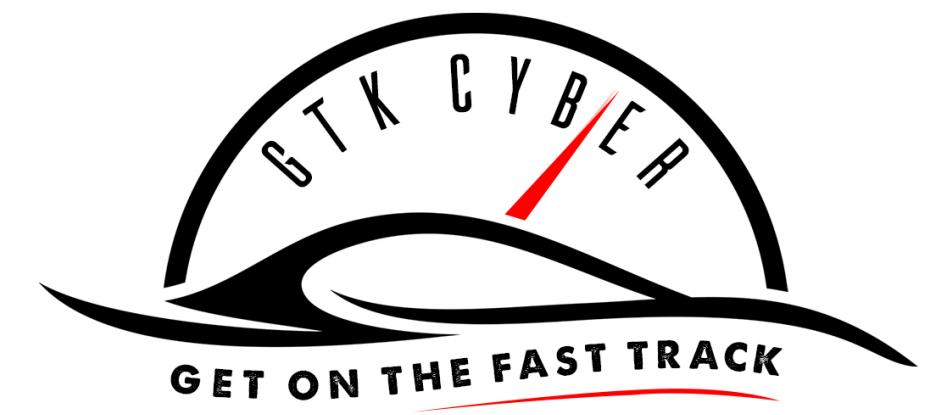
YAHOO!



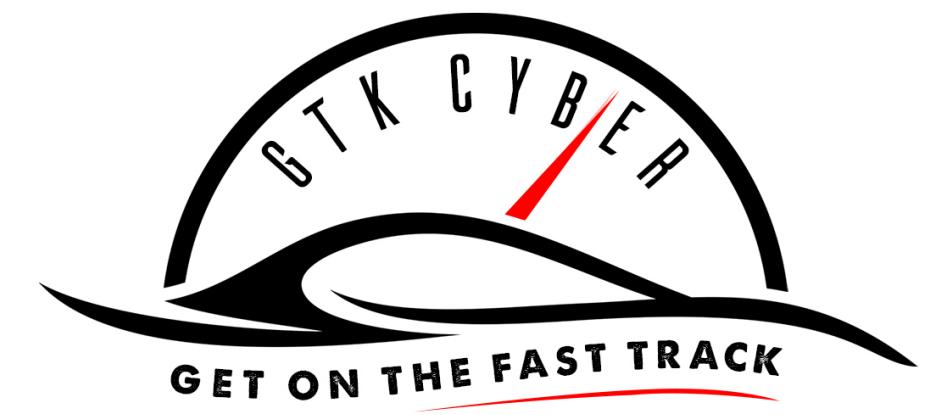
```
input_lines = LOAD '/tmp/my-copy-of-all-pages-on-internet' AS  
(line:chararray);  
  
-- Extract words from each line and put them into a pig bag  
-- datatype, then flatten the bag to get one word on each row  
words = FOREACH input_lines GENERATE FLATTEN(TOKENIZE(line))  
AS word;  
  
-- filter out any words that are just white spaces  
filtered_words = FILTER words BY word MATCHES '\\\\w+';  
  
-- create a group for each word  
word_groups = GROUP filtered_words BY word;  
  
-- count the entries in each group  
word_count = FOREACH word_groups GENERATE  
COUNT(filtered_words) AS count, group AS word;  
  
-- order the records by count  
ordered_word_count = ORDER word_count BY count DESC;  
STORE ordered_word_count INTO '/tmp/number-of-words-on-  
internet';
```

[https://en.wikipedia.org/wiki/Pig_\(programming_tool\)](https://en.wikipedia.org/wiki/Pig_(programming_tool))

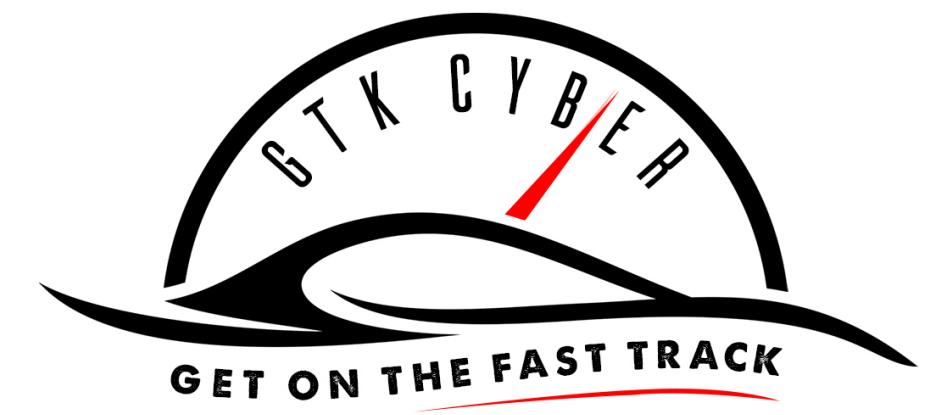




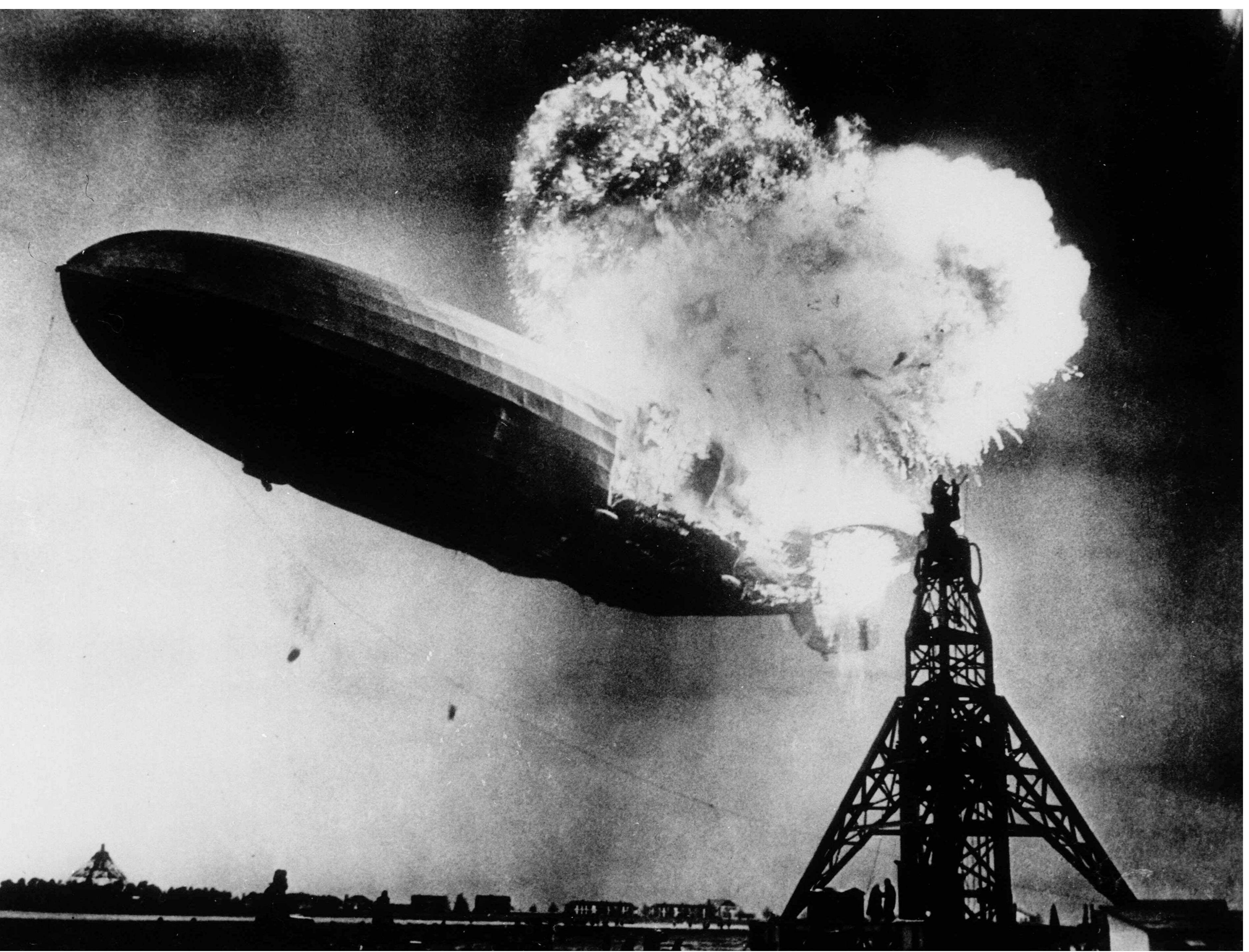
```
SELECT word, count(1) as count
FROM
(
    SELECT explode(split(sentence, ' ')) AS word
    FROM texttable
) tempTable
GROUP BY word
```

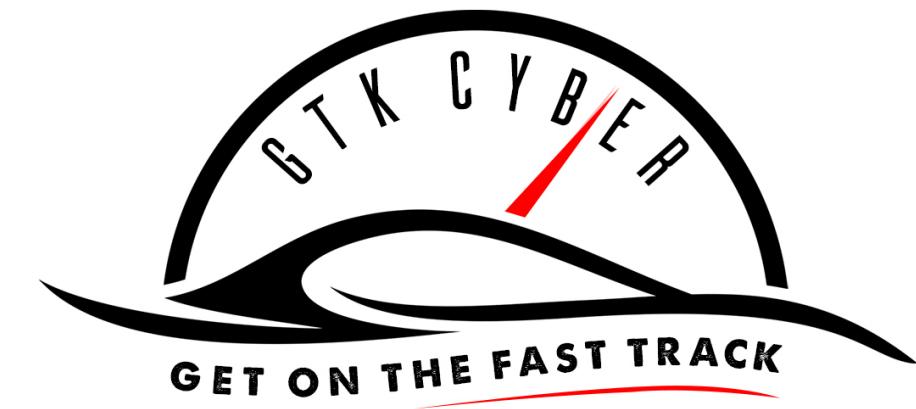


Apache Zeppelin

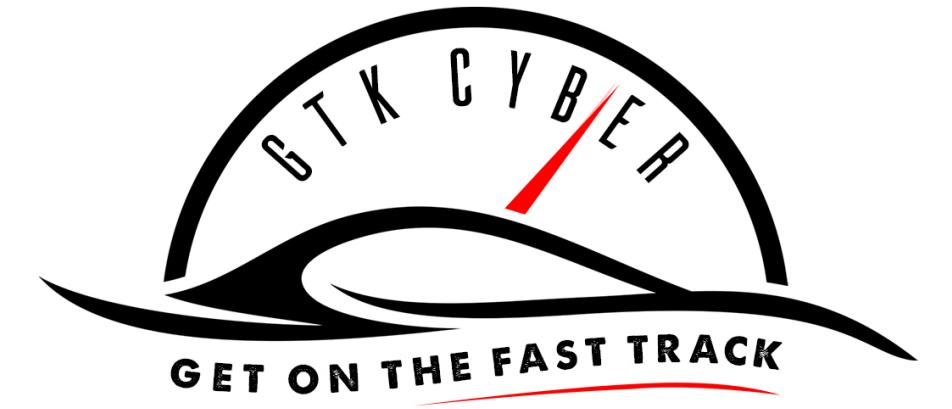


What is it?

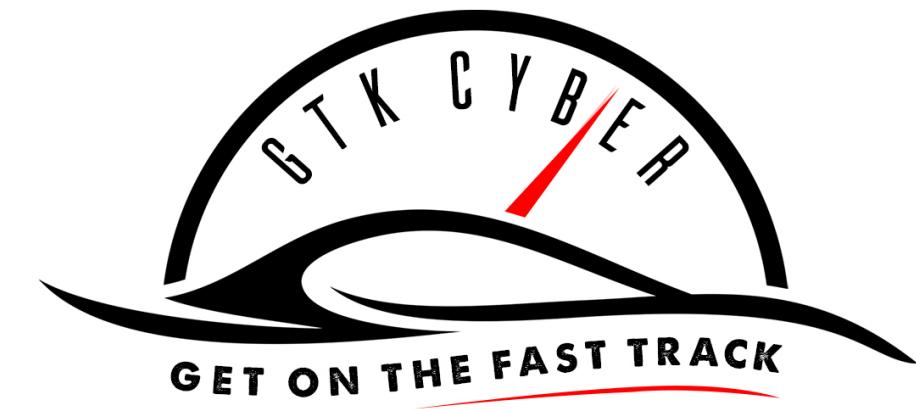




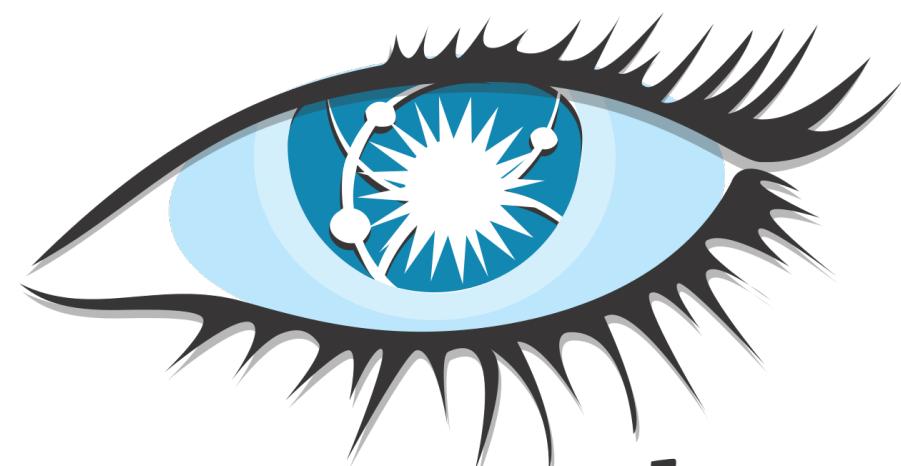
A web based notebook
that enables interactive
data analytics



Built in integration
with 
spark



GEODE



cassandra

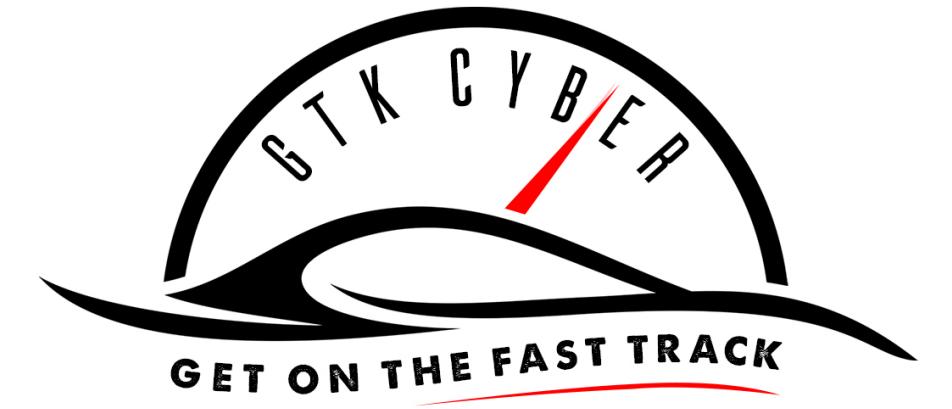


PostgreSQL

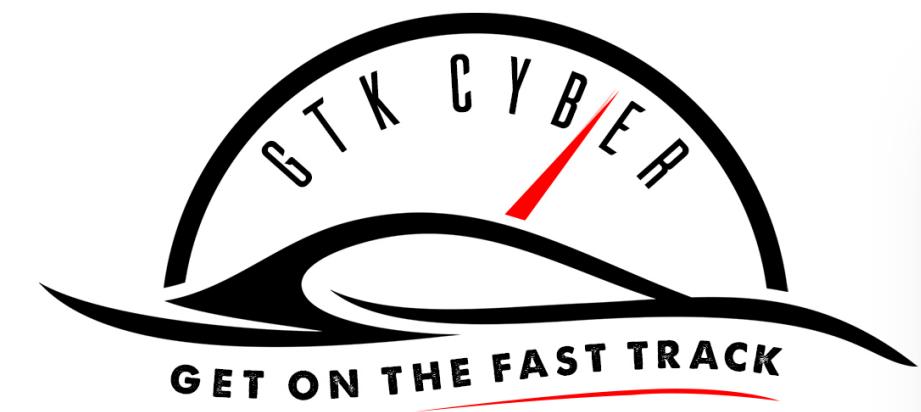


Flink





Zeppelin makes
exploring data easy



localhost

Connected

Welcome to Zeppelin.

This is a live tutorial, you can run the code yourself. (Shift-Enter to Run)

Took 0 seconds

FINISHED

```
%sql
select age, count(1) value
from bank
where age < 30
group by age
order by age
```

FINISHED

```
%sql
select age, count(1) value
from bank
where age < ${maxAge=30}
group by age
order by age
```

maxA 40

FINISHED

```
%sql
select age, count(1) value
from bank
where marital="${marital=single, single|divorced|married}"
group by age
```

marital singl

Grouped Stacked value

103.0
80.0
60.0
40.0
20.0
0.0

23 28

Took 1 seconds

Grouped Stacked value

2,085.0
1,500.0
1,000.0
500.0
0.0

27 37

Took 0 seconds

Grouped Stacked value

1,017.0
800.0
600.0
400.0
200.0
0.0

30 42 70

Took 0 seconds

cmd

FINISHED

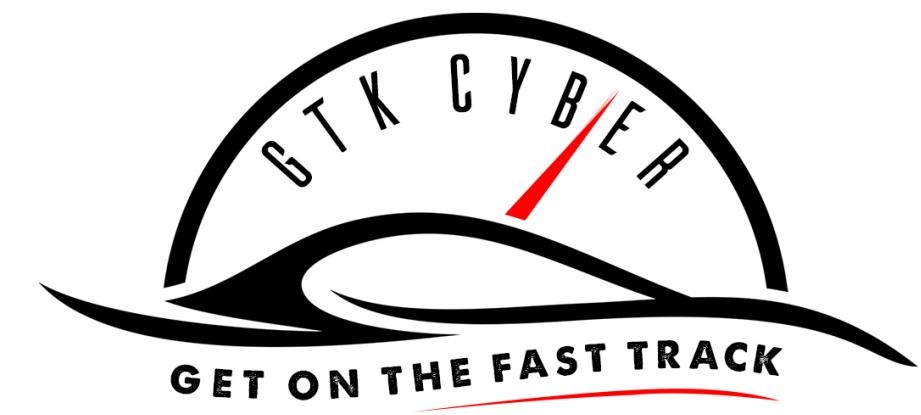


```
val bankText = sc.textFile("/Users/cgivre/Downloads/bank/bank-full.csv")

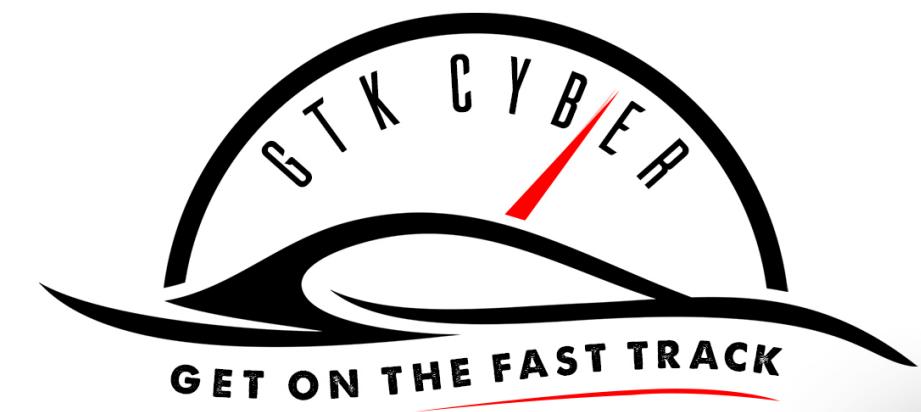
case class Bank(age:Integer, job:String, marital : String, education : String, ba
Integer)

val bank = bankText.map(s=>s.split(";")).filter(s=>s(0) != "age\").map(
s=>Bank(s(0).toInt,
        s(1).replaceAll("\\"", ""),
        s(2).replaceAll("\\"", ""),
        s(3).replaceAll("\\"", ""),
        s(5).replaceAll("\\"", "").toInt
)
)

bank.toDF().registerTempTable("bank")
```



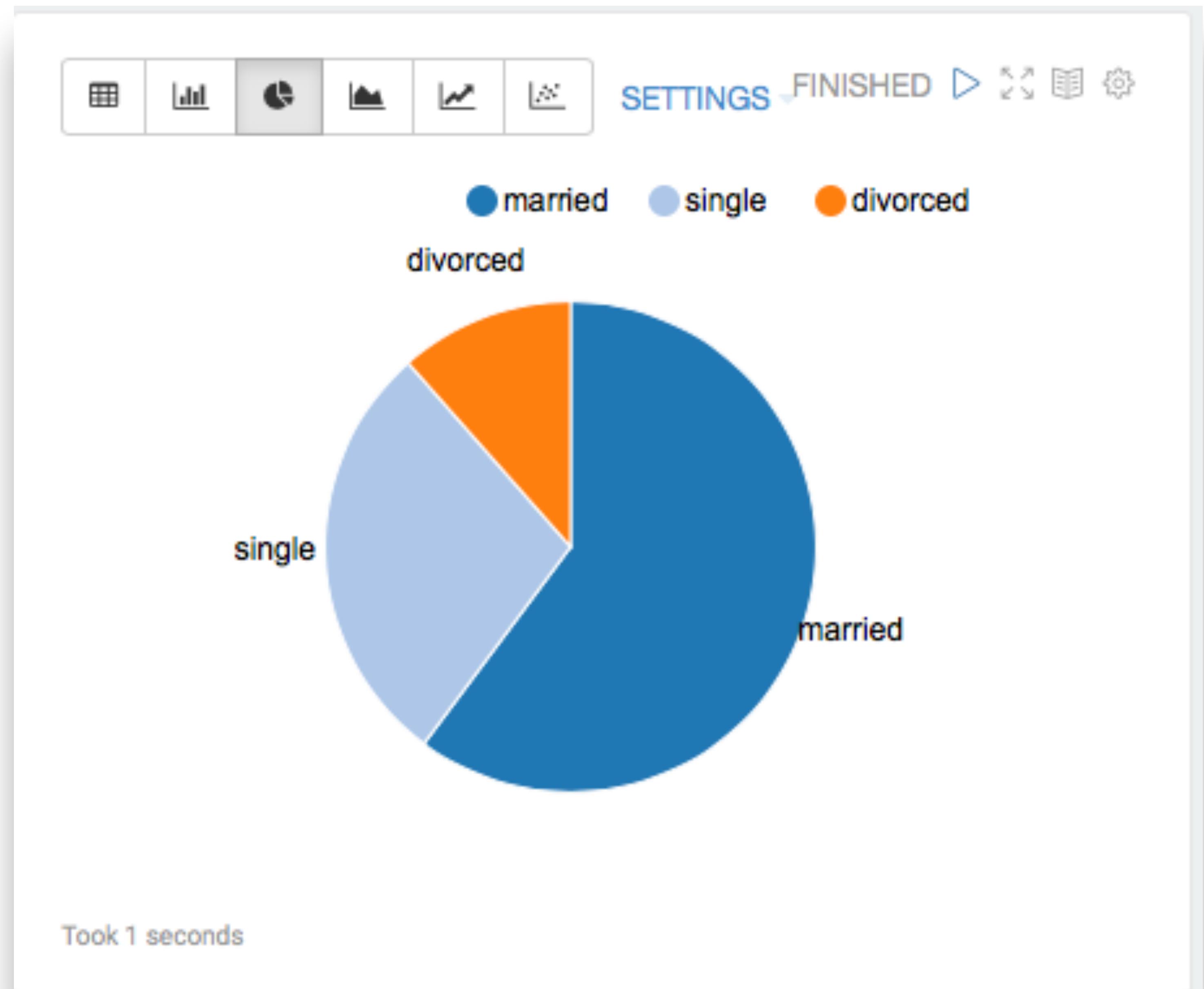
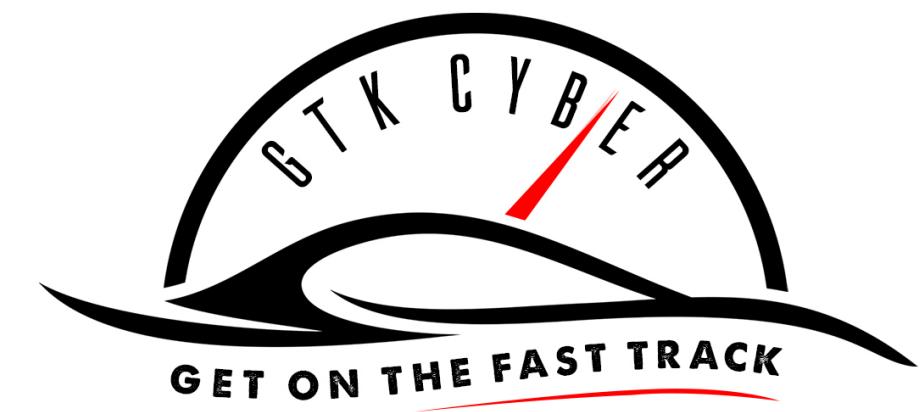
```
%sql  
SELECT marital, COUNT( 1 ) AS statusCount  
FROM bank  
GROUP BY marital  
ORDER BY statusCount DESC
```

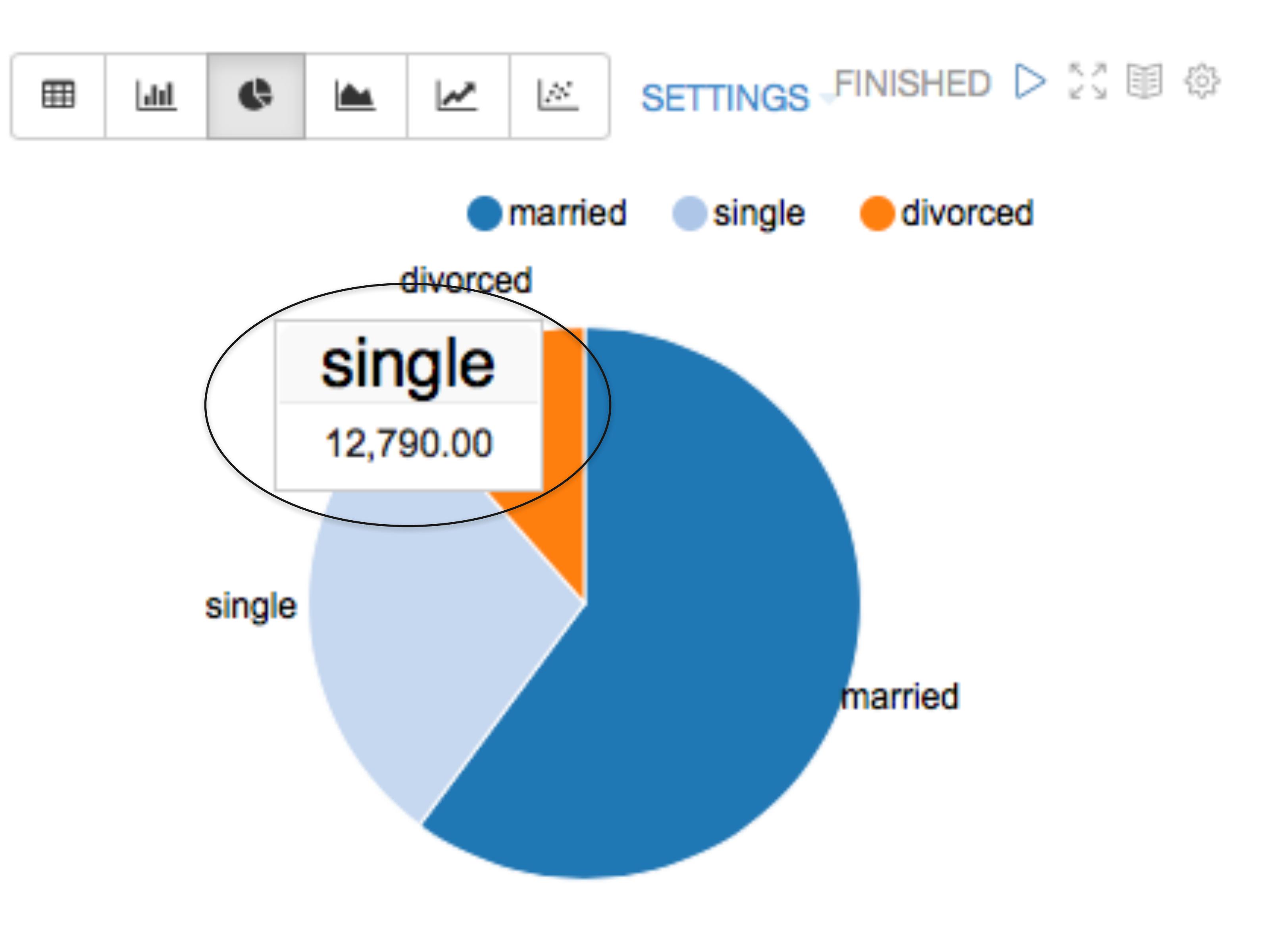
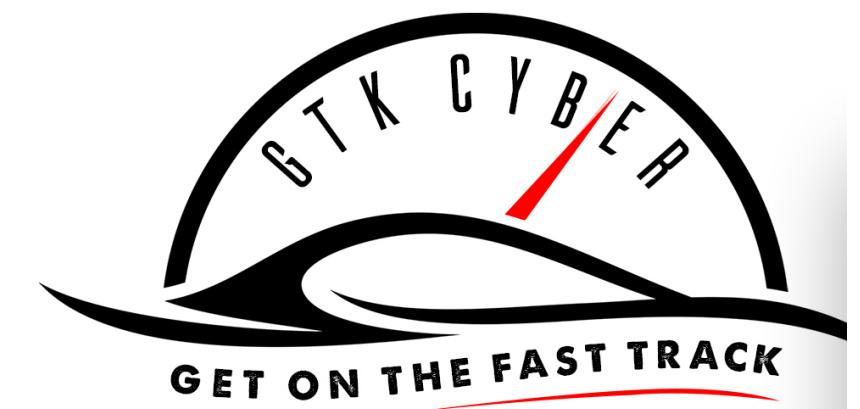


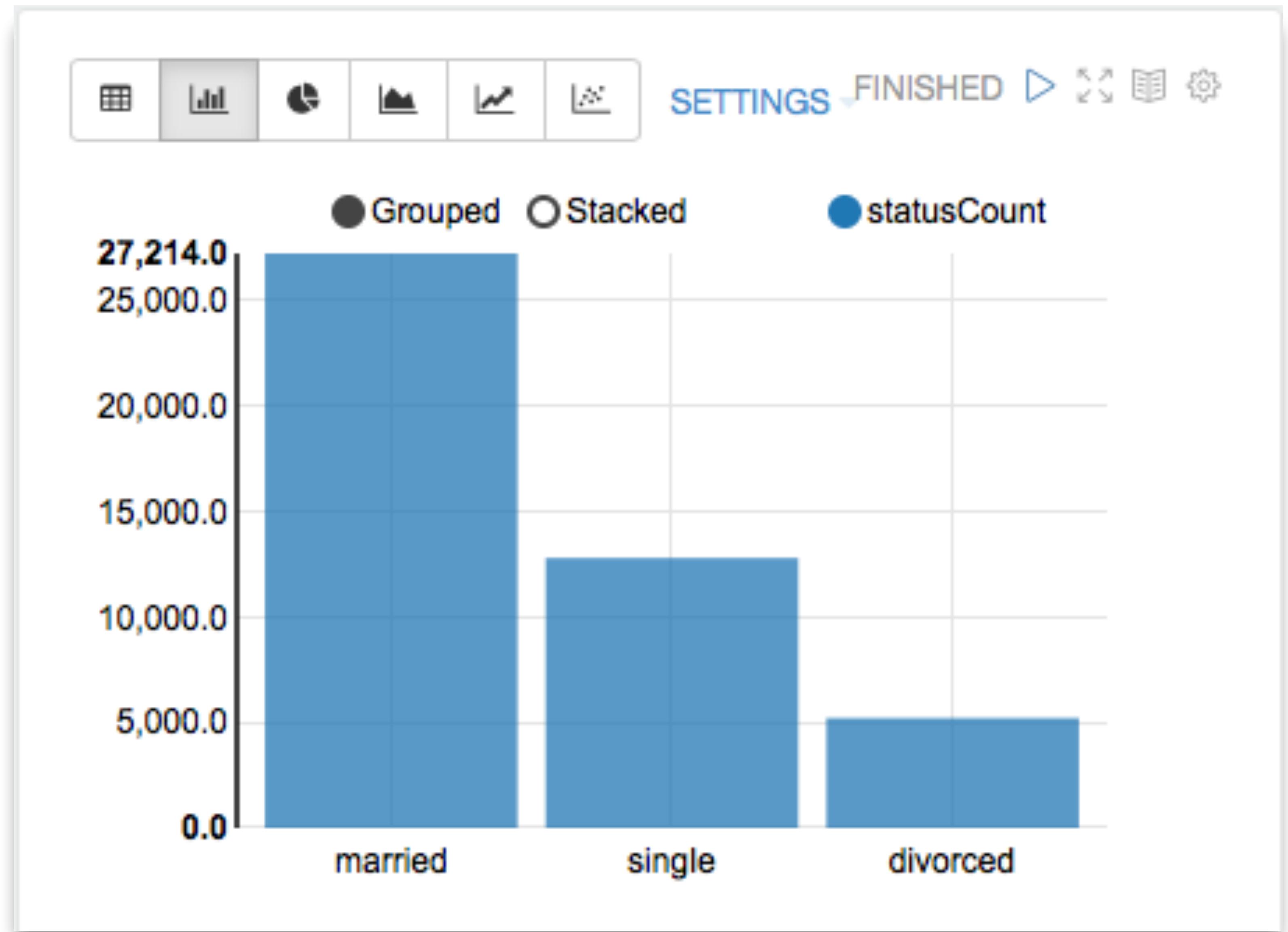
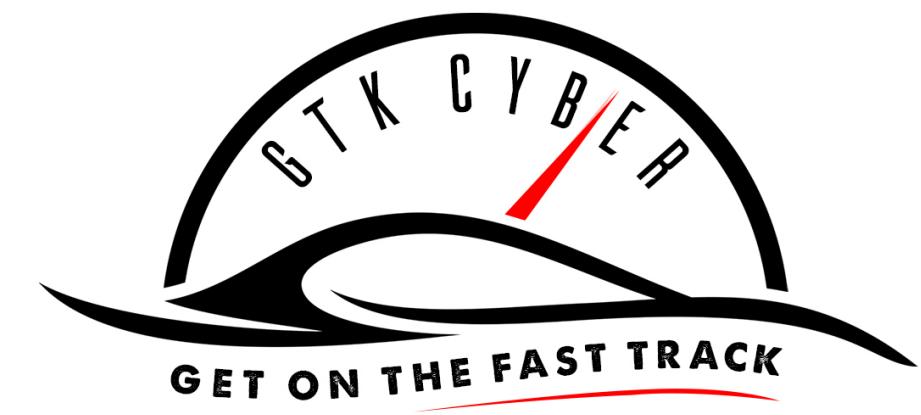
```
%sql
SELECT marital, COUNT(1) AS statusCount
FROM bank
GROUP BY marital
ORDER BY statusCount DESC
```

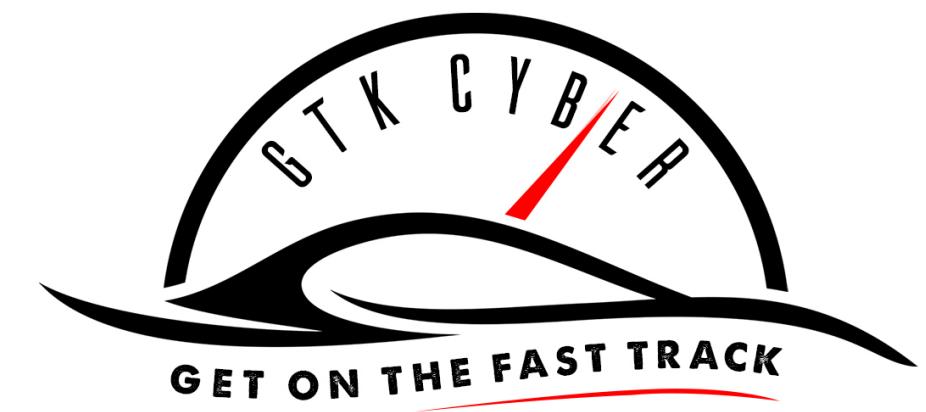
The screenshot shows a SQL query results interface. At the top, the command '%sql' is entered, followed by the query. To the right, the status 'FINISHED' is displayed along with several icons for navigation and settings. Below the query, there is a toolbar with six icons: a grid (selected), a bar chart, a pie chart, a line chart, and two other charts. A large oval highlights this toolbar area. The main content area displays a table with two columns: 'marital' and 'statusCount'. The data rows are: 'married' with a count of 27,214, 'single' with a count of 12,790, and 'divorced' with a count of 5,207.

marital	statusCount
married	27,214
single	12,790
divorced	5,207

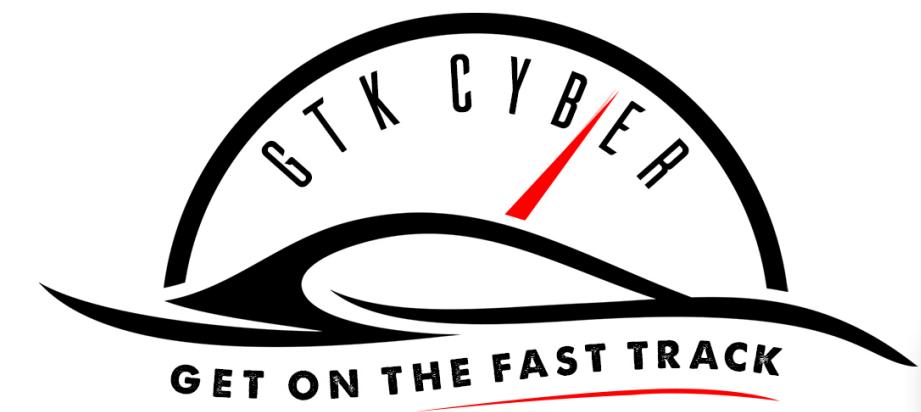








```
%sql
SELECT education, age, AVG( balance) as avgBalance
FROM bank
WHERE age > 20 AND age < 35
GROUP BY age, education
ORDER BY age ASC
```



SETTINGS FINISHED ▶ 🔍 📄 ⚙

All fields:

education age avgBalance

Keys

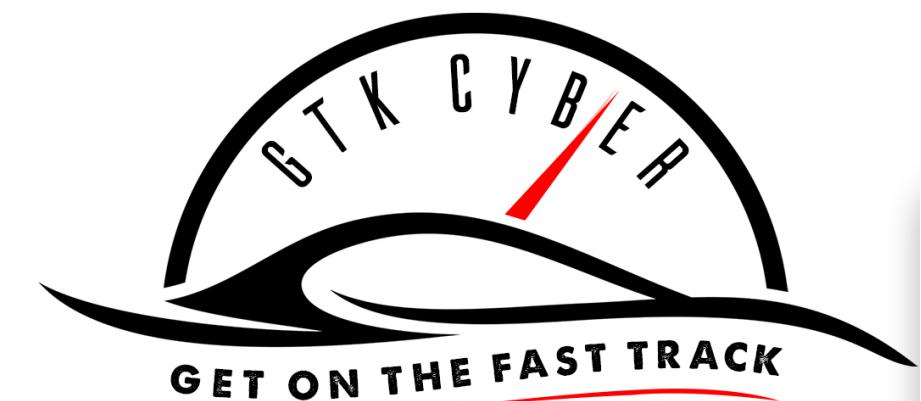
age ✖

Groups

education ✖

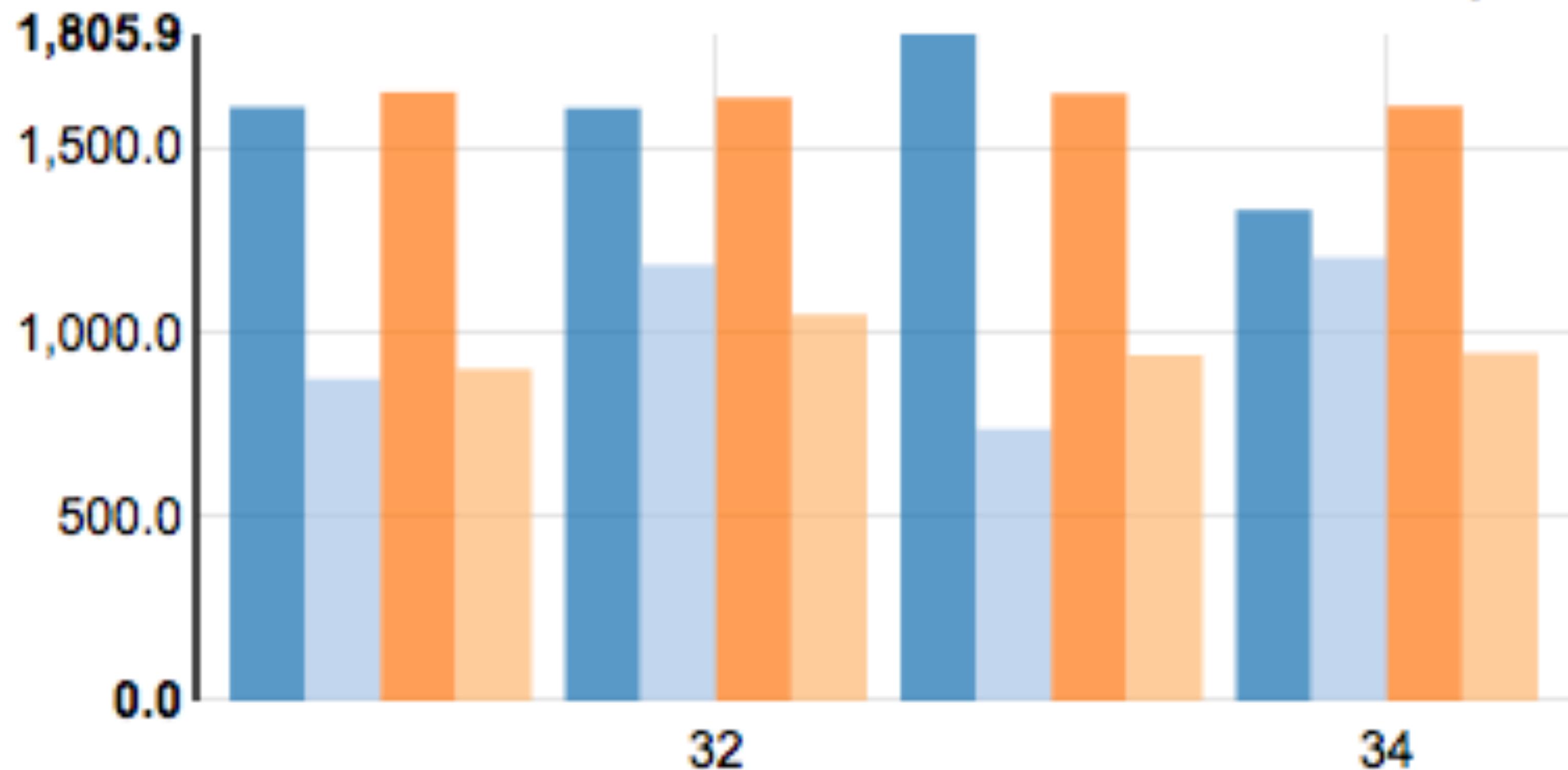
Values

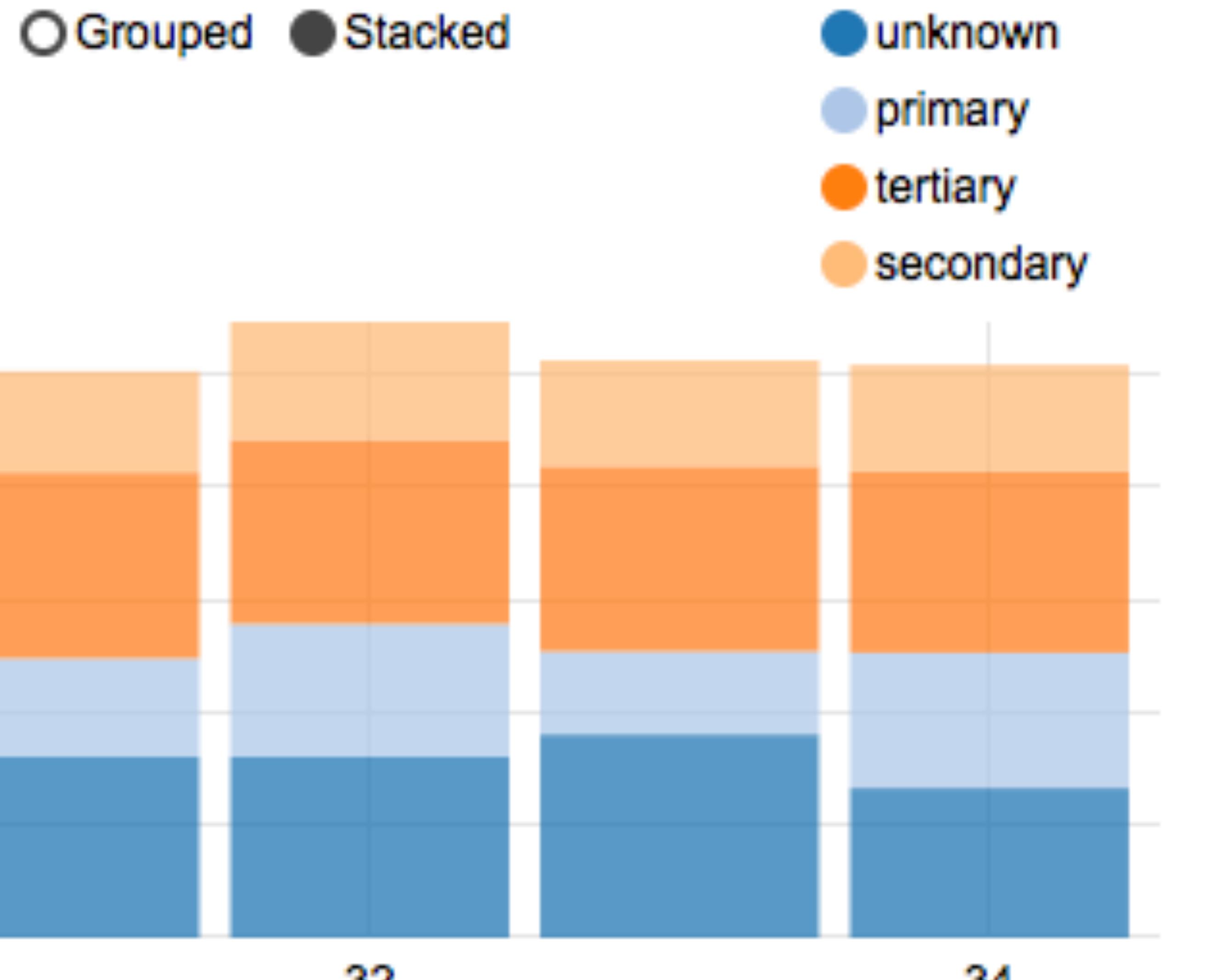
avgBalance SUM ✖

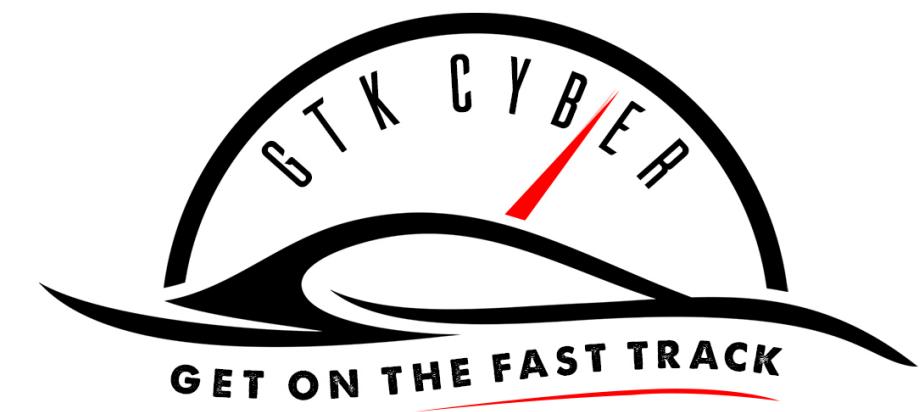


● Grouped ○ Stacked

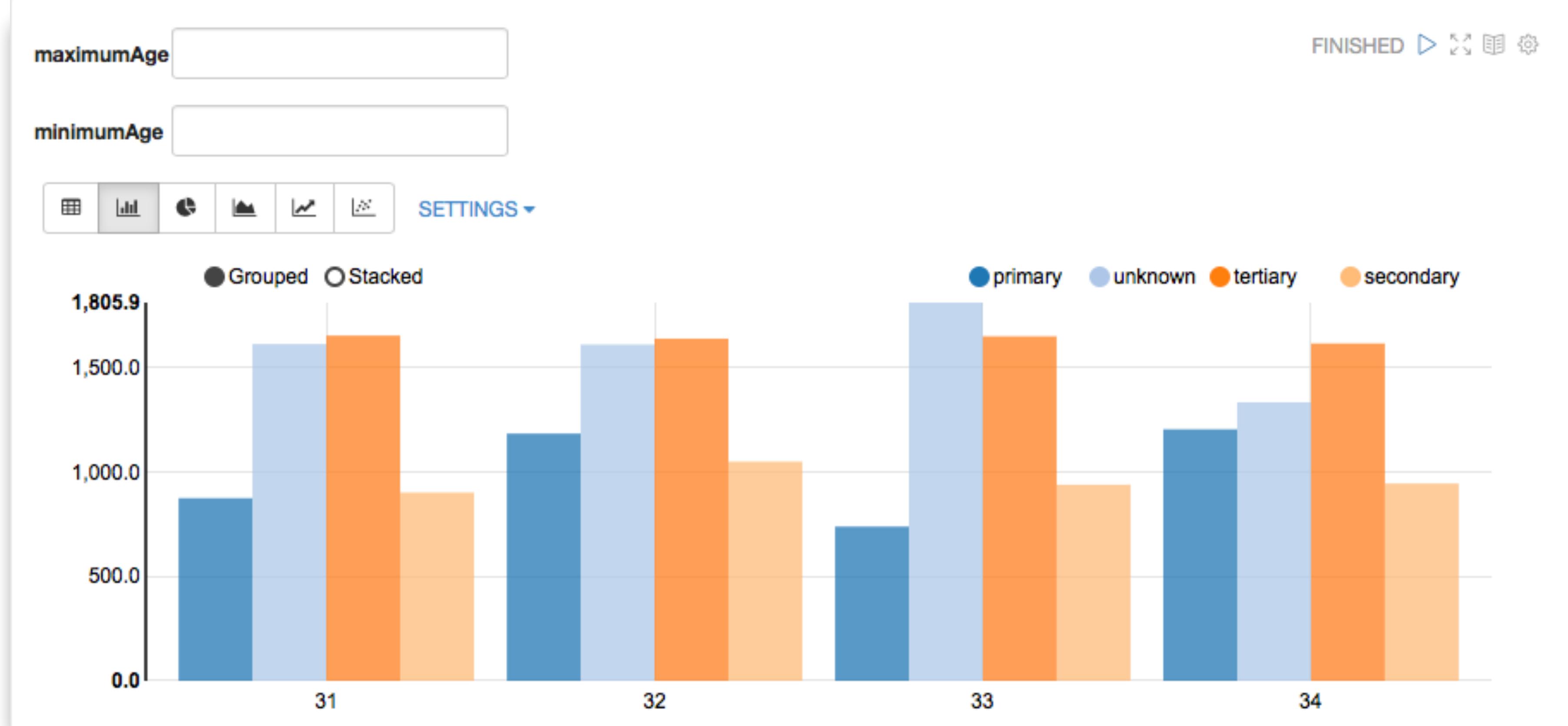
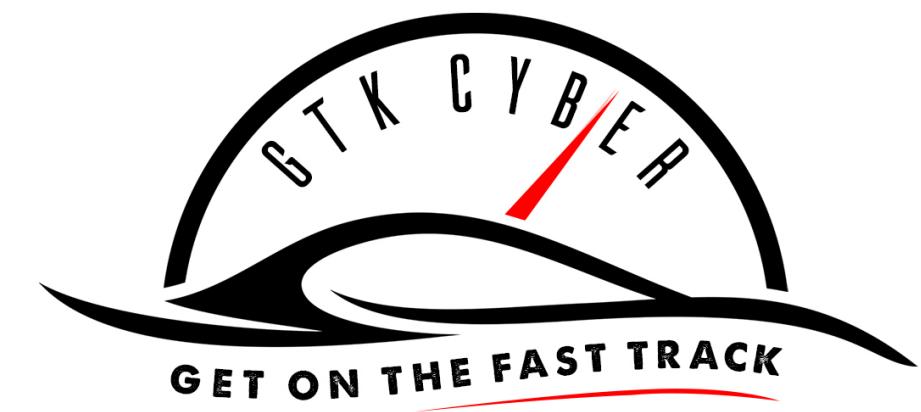
- unknown
- primary
- tertiary
- secondary

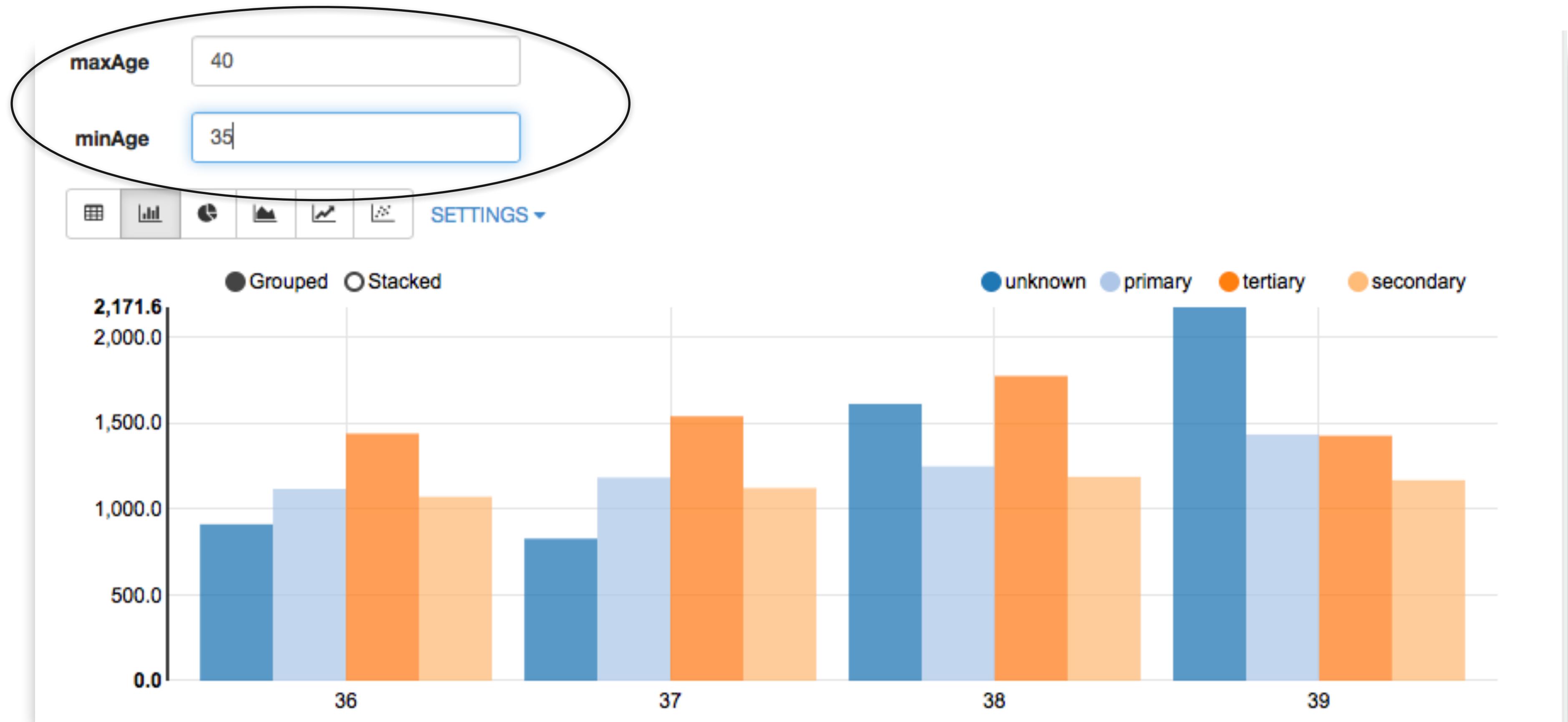
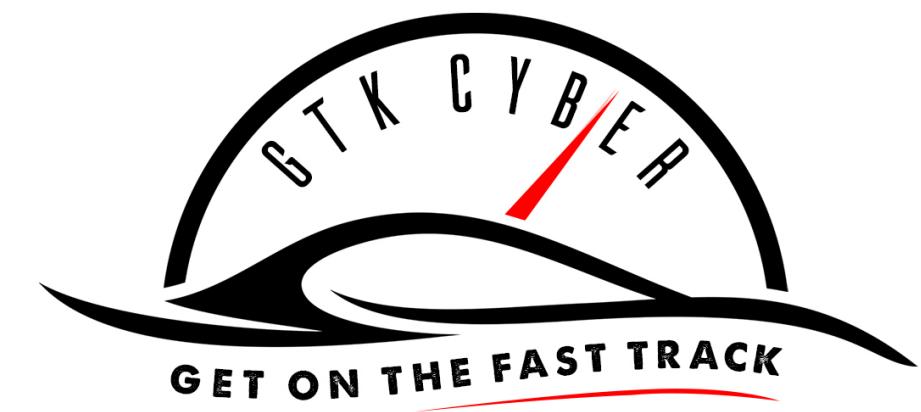


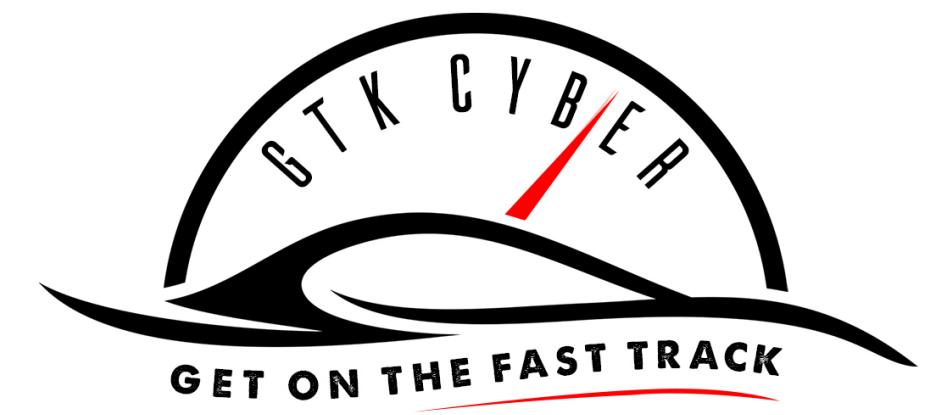




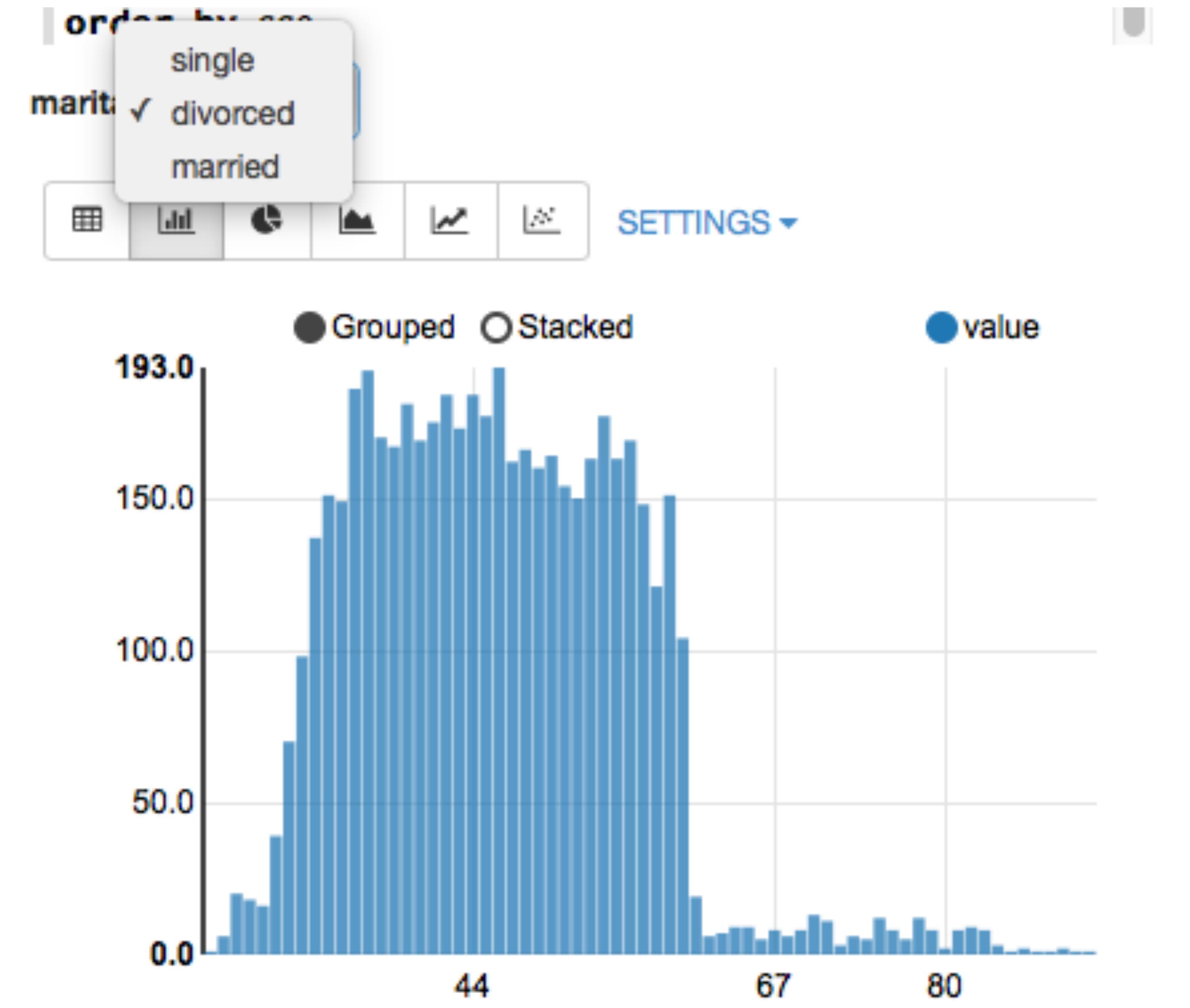
```
%sql
SELECT education, age, AVG( balance) as
avgBalance
FROM bank
WHERE age > ${minimumAge=30} AND age < ${
maximumAge=35}
GROUP BY age, education
ORDER BY age ASC
```

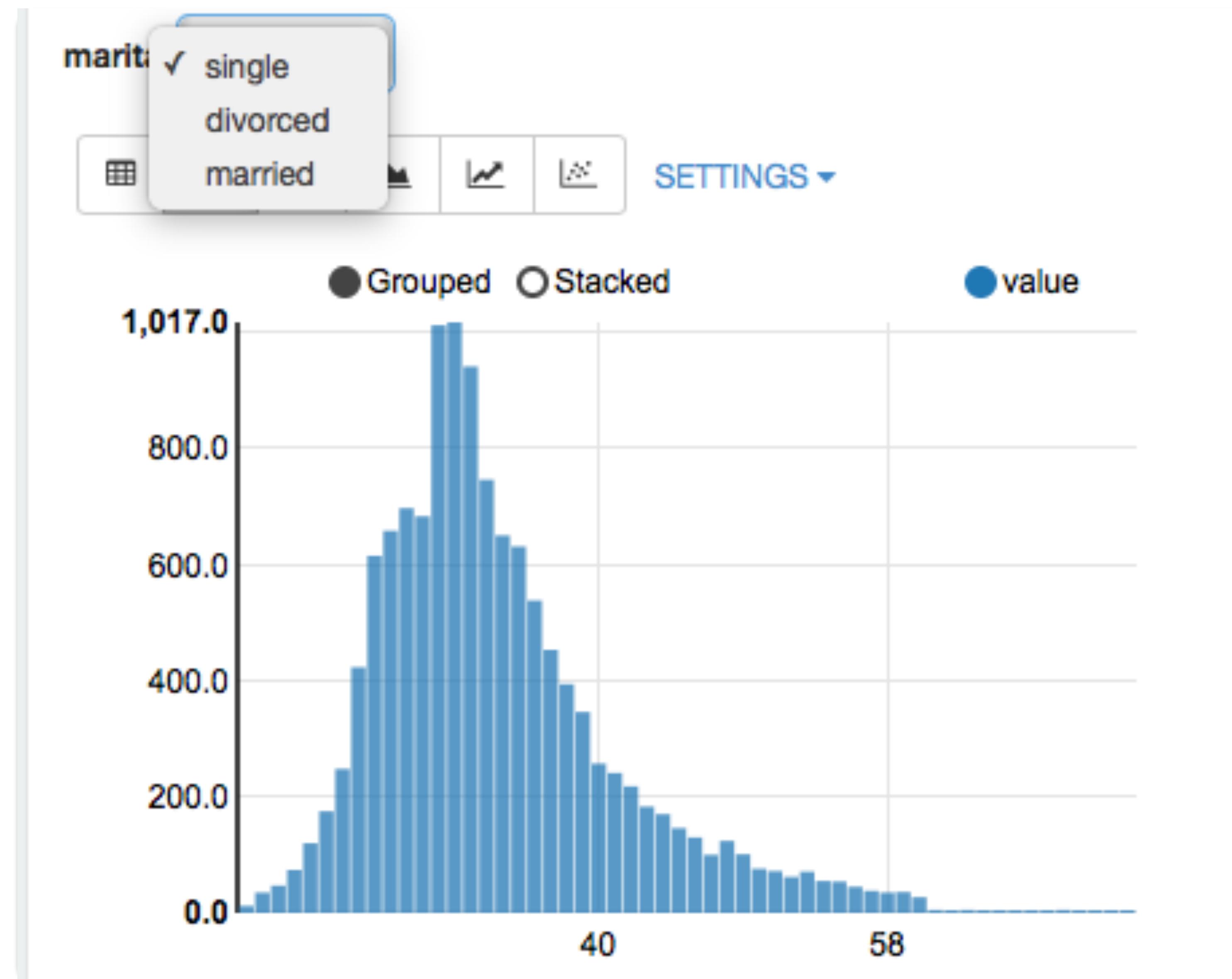


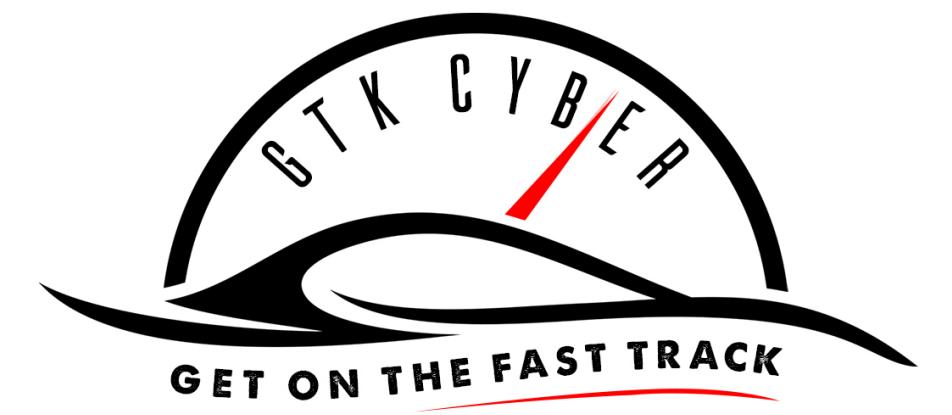




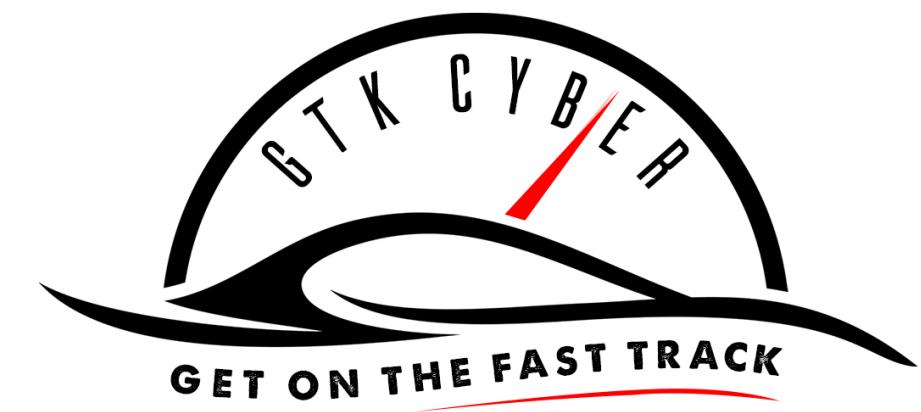
```
%sql
SELECT age, count(1) value
FROM bank
WHERE marital="${marital=single,single|divorced|married}"
GROUP BY age
ORDER BY age
```







Importing a notebook



Importing a notebook

The screenshot shows a Mozilla Firefox window with the title "Griffon RC4 [Running]". The address bar displays "http://localhost:8060/#/". The main content area is the Zeppelin web interface, featuring a blue header with the Zeppelin logo, a search bar, and a user status indicator ("anonymous"). The main page displays a large "Welcome to Zeppelin!" heading and a brief introduction about the tool's purpose. On the left, there is a sidebar with options like "Notebook", "Import note" (which is circled in red), "Create new note", "Filter", and links to "Apache Drill Demo", "Zeppelin Tutorial", and "Trash". On the right, there are sections for "Help", "Community", and links to "Mailing list", "Issues tracking", and "Github".

Griffon RC4 [Running]

Applications Places System Jupyter Nb R E

Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://localhost:8060/ #

Search

Most Visited Getting Started

Zeppelin Notebook Job Search your Notes anonymous

Welcome to Zeppelin!

Zeppelin is web-based notebook that enables interactive data analytics. You can make beautiful data-driven, interactive, collaborative document with SQL, code and even more!

Notebook

Import note

Create new note

Filter

Apache Drill Demo

Zeppelin Tutorial

Trash

Help

Get started with [Zeppelin documentation](#)

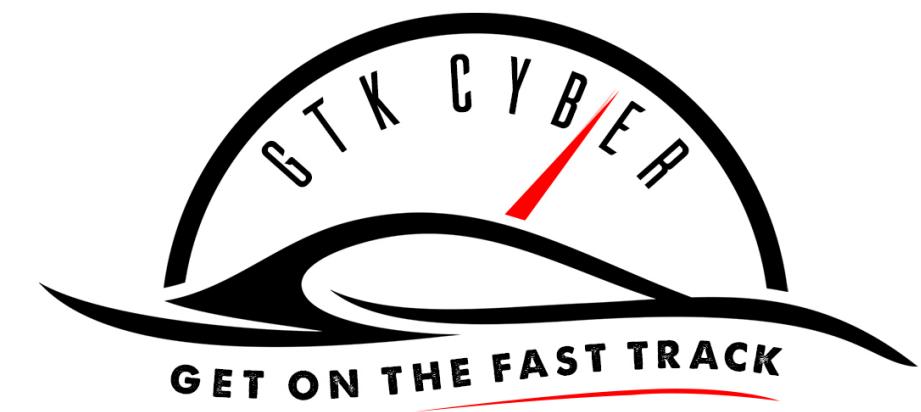
Community

Please feel free to help us to improve Zeppelin, Any contribution are welcome!

Mailing list

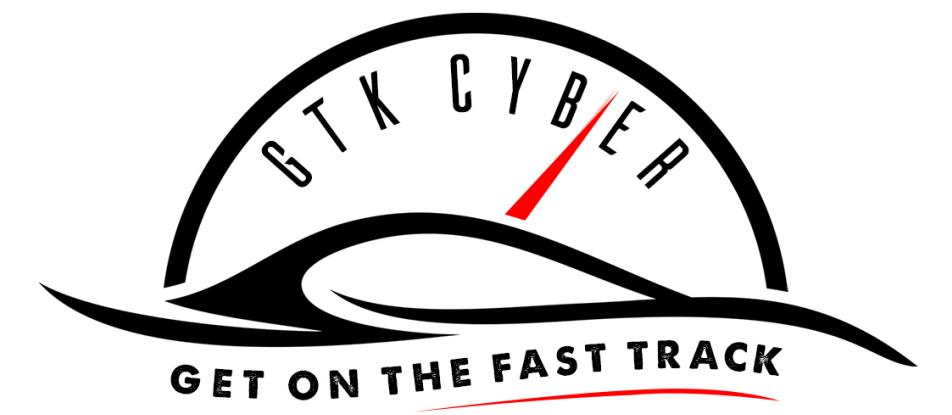
Issues tracking

Github

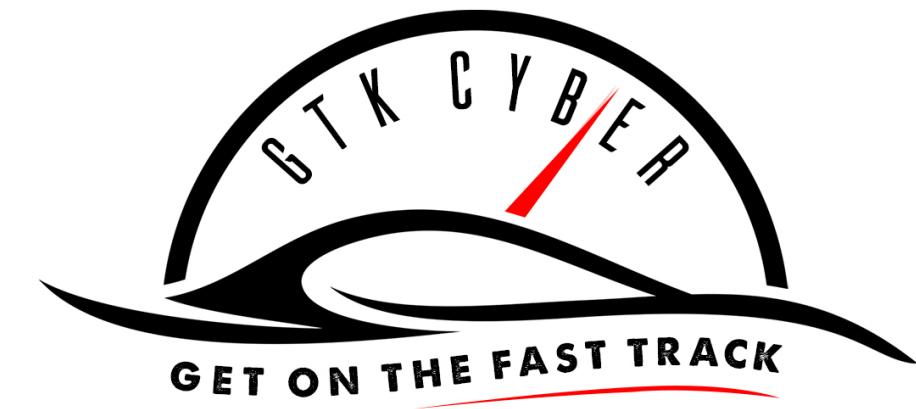


Importing a notebook

The screenshot shows a Mozilla Firefox window running on a system named "Griffon RC4 [Running]". The address bar shows the URL <http://localhost:8060/>. The main content area is the Zeppelin web interface. A modal dialog box is centered over the page, titled "Import new note". Inside the dialog, there is a field labeled "Import AS" with a dropdown menu containing "Note name". Below this, a warning message states "JSON file size cannot exceed 1 MB". There are two main import options: "Choose a JSON here" represented by a cloud icon with an upward arrow, and "Add from URL" represented by a chain-link icon. The background of the browser shows the Zeppelin dashboard with sections for "Welcome to", "Notebook", and "Job".

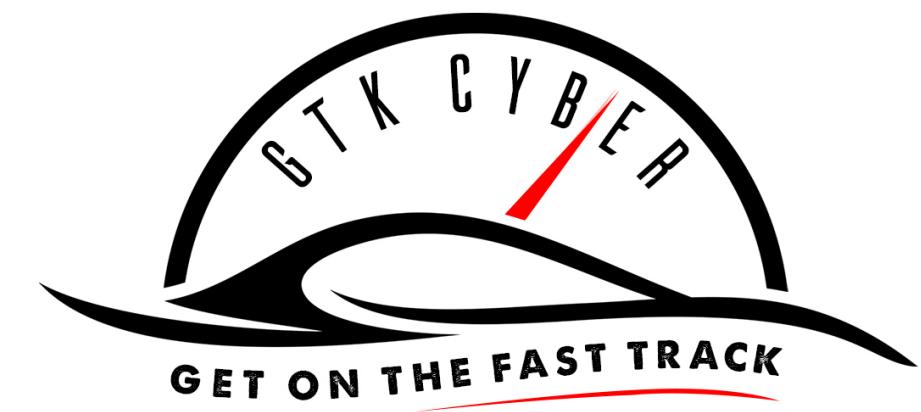


spark³

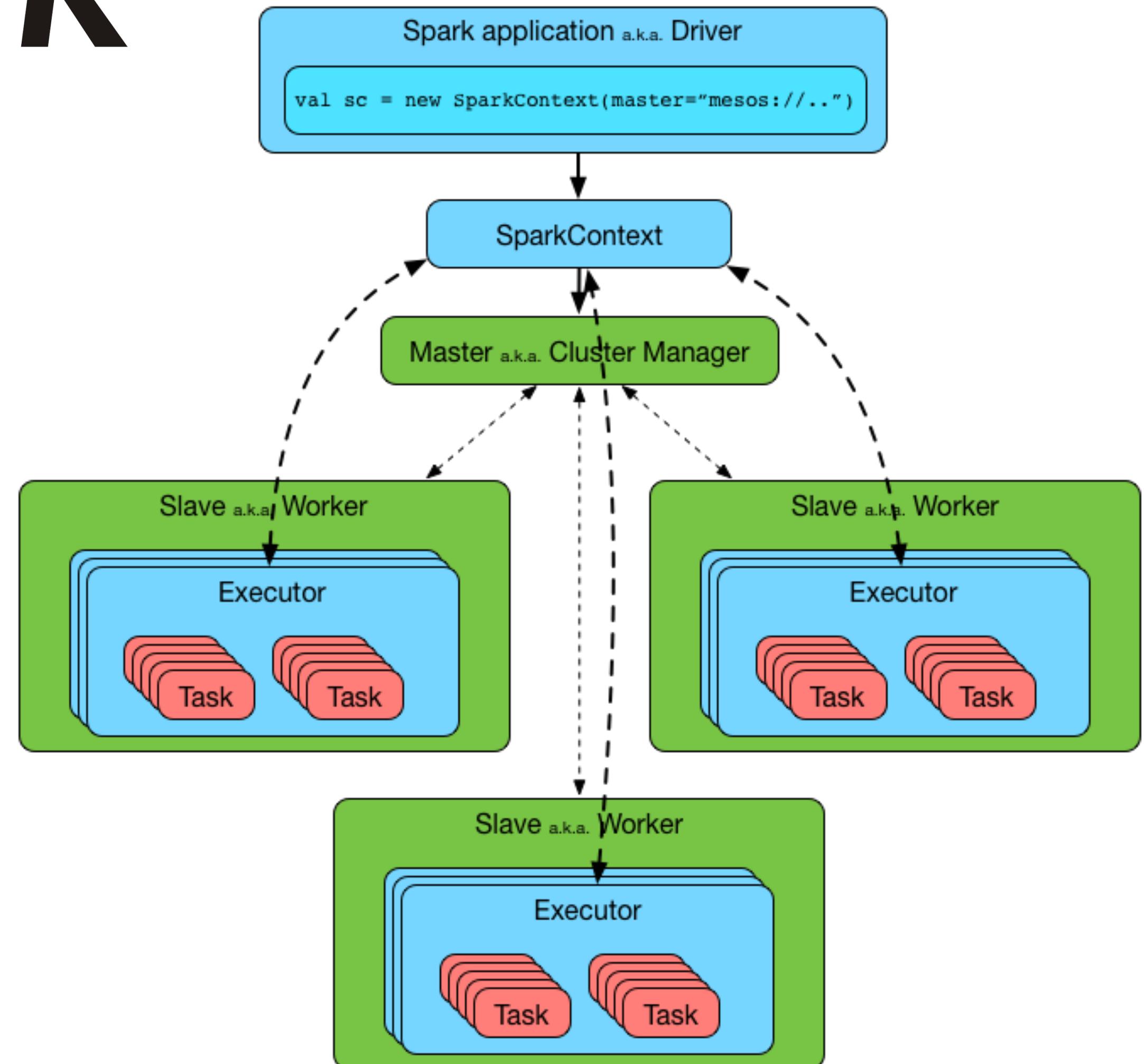
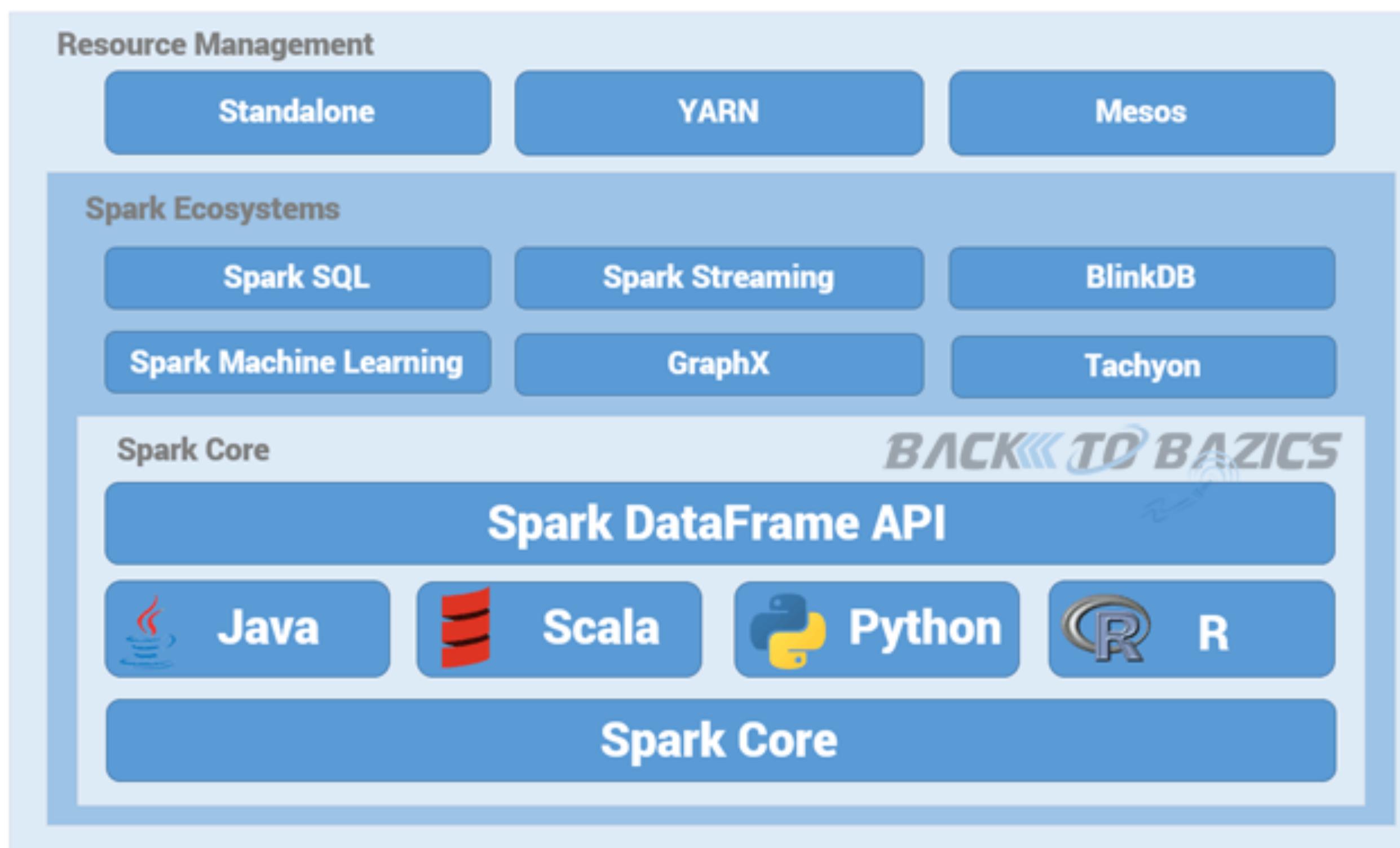


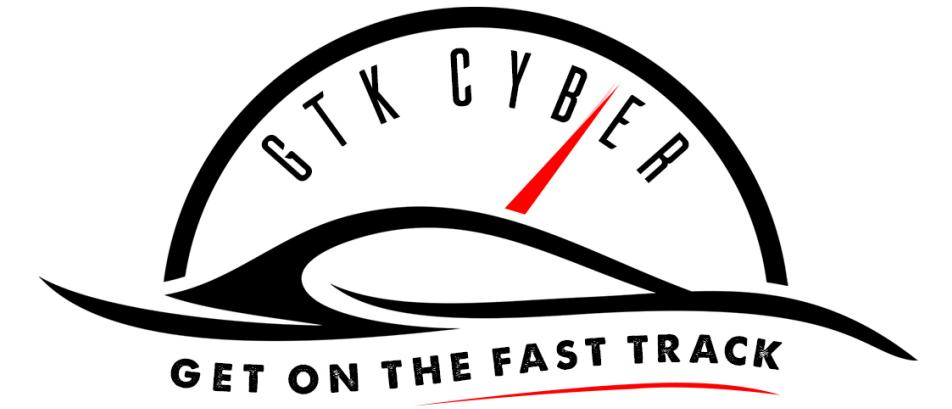
Apache Spark™ is a fast and general engine for large-scale data processing.

- **Speed:** Run programs up to 100x faster than Hadoop MapReduce **in memory**, or 10x faster on disk.
- **Ease of Use:** Write applications quickly in Java, Scala, Python via **PySpark**, R.
- **Generality:** Combine SQL, **streaming**, and complex analytics.
- **Runs Everywhere:** Spark runs on Hadoop, Mesos, standalone, or in the cloud. It can access diverse data sources including HDFS, Cassandra, HBase, and S3.

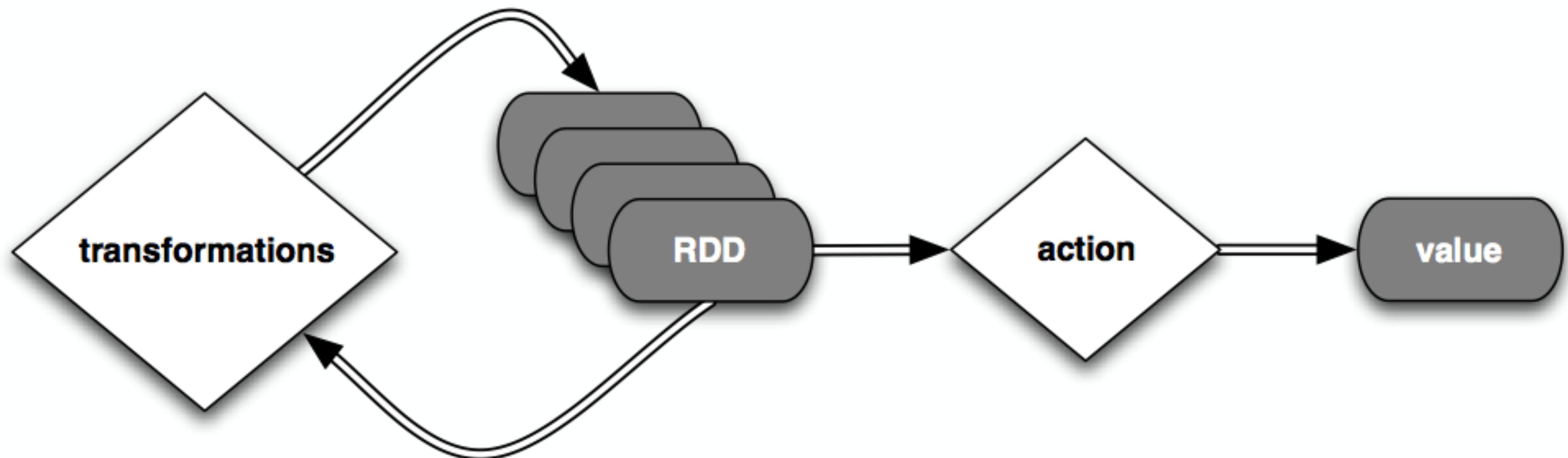
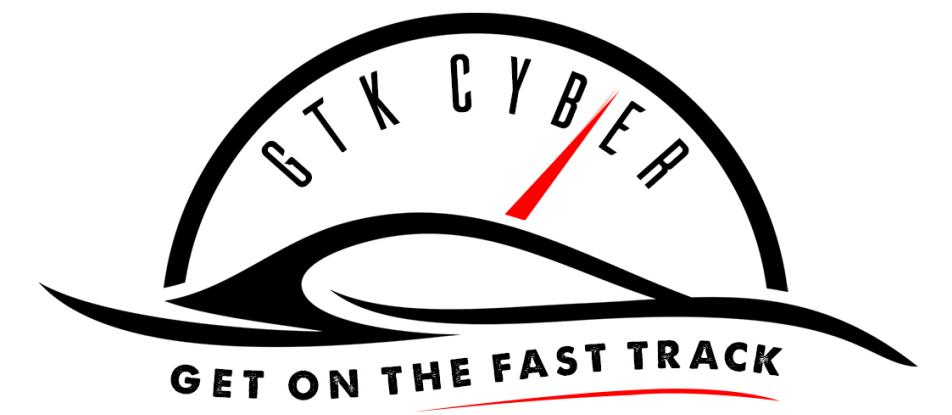


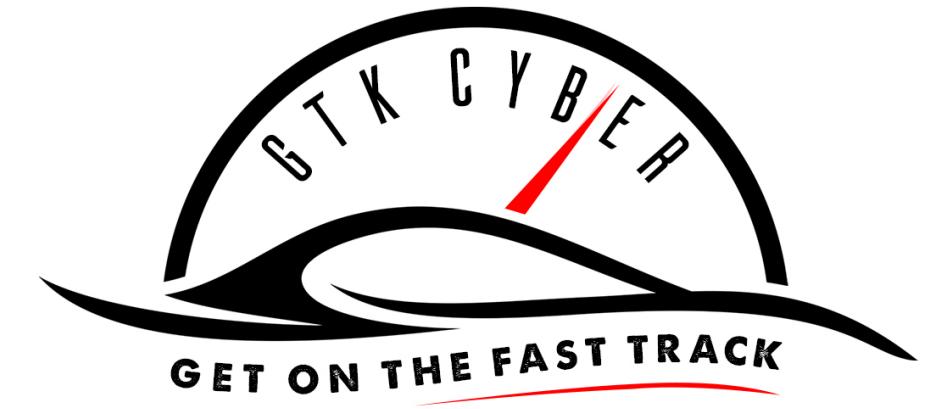
Spark



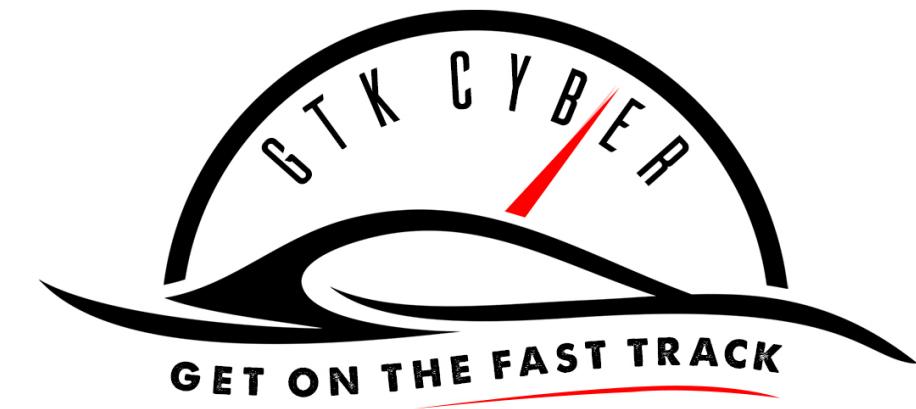


Spark has two classes of operations: transformations, and actions.





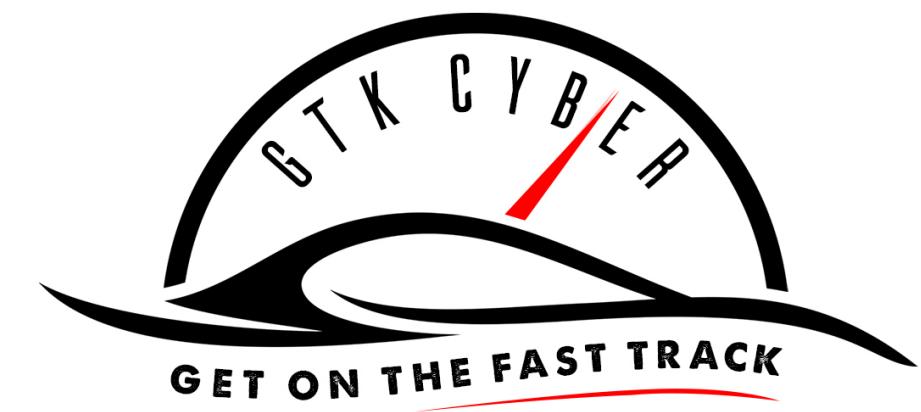
rdd - Resilient Distributed Dataset



```
from operator import add

f = sc.textFile("README.md")
wc = f.flatMap( lambda x: x.split(' '))
    .map( lambda x: (x, 1) )
    .reduceByKey(add)

wc.saveAsTextFile( "wc_out.txt" )
```



```
from pyspark.sql import SQLContext
from pyspark import SparkContext
sc = SparkContext()

sqlCtx = SQLContext(sc)

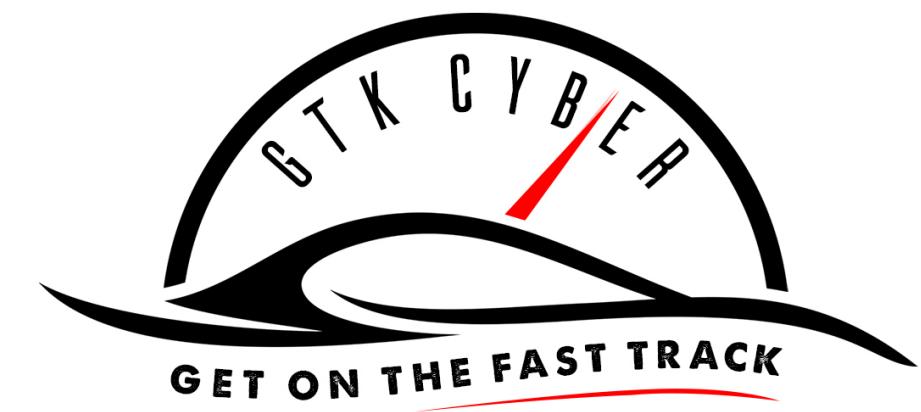
#Load a text file and convert each line to a dictionary
lines = sc.textFile("examples/src/main/resources/people.txt")

parts = lines.map(lambda l: l.split(","))
people = parts.map(lambda p: {"name": p[0], "age": int(p[1])})

peopleTable = sqlCtx.inferSchema(people)
peopleTable.registerAsTable("people")

# SQL can be run over SchemaRDDs that have been registered as a table
teenagers = sqlCtx.sql("SELECT name FROM people WHERE age >= 13 AND age <= 19")

teenNames = teenagers.map(lambda p: "Name: " + p.name)
teenNames.collect()
```



Parse Apache Access Logs using RDD map

```
# Version 1: Parsing of Apache Logs via RDD
regex_pattern = '^(\\S+) (\\S+) (\\S+) \\[(\\[\\w:/]+\\s[+\\-]\\d{4})\\] "(\\S+) (\\S+) (\\S+)" (\\d{3}) (\\d+)'

def map_apache_log(line):
    match = re.search(regex_pattern, line)
    if match is None:
        log_row = Row(
            Host='null',
            ClientID='null',
            UserID='null',
            DateTime='null',
            Method='null',
            Endpoint='null',
            Protocol='null',
            Response='null',
            SizeBytes='null')
    else:
        log_row = Row(
            Host=match.group(1),
            ClientID=match.group(2),
            UserID=match.group(3),
            DateTime=match.group(4),
            Method=match.group(5),
            Endpoint=match.group(6),
            Protocol=match.group(7),
            Response=match.group(8),
            SizeBytes=match.group(9))
    return log_row

%pyspark
DATA_HOME = '/home/griffonuser/applied-ds/data/'
fname = DATA_HOME + 'apache-access.log'

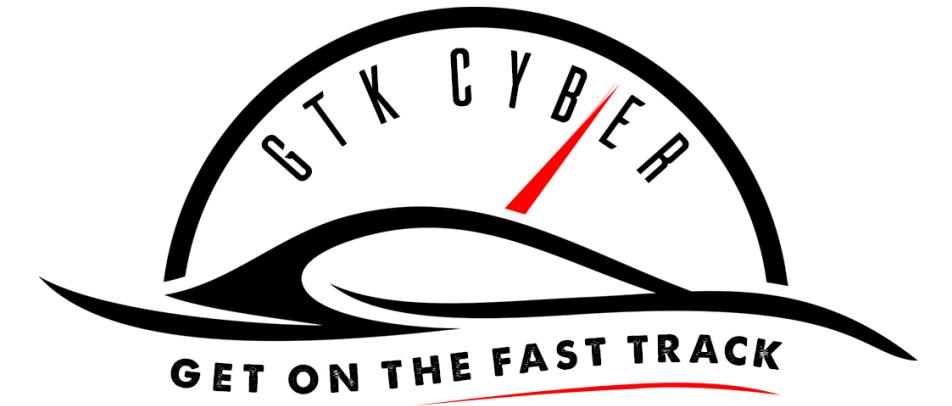
# read rdd, map each row (see function above) and convert to DataFrame
rdd = sc.textFile(fname).map(map_apache_log)
df_from_rdd = rdd.toDF()
df_from_rdd = df_from_rdd.na.drop(how='any')

# Define data types
to_datetime = udf(lambda x: datetime.strptime(x, '%d/%b/%Y:%H:%M:%S %z'), TimestampType())

df_from_rdd = df_from_rdd.withColumn("Response", df_from_rdd.Response.cast(IntegerType())) \
    .withColumn("SizeBytes", df_from_rdd.SizeBytes.cast(IntegerType()))
df_from_rdd = df_from_rdd.na.drop(how='any')
df_from_rdd = df_from_rdd.withColumn("DateTime", to_datetime(col("DateTime")))

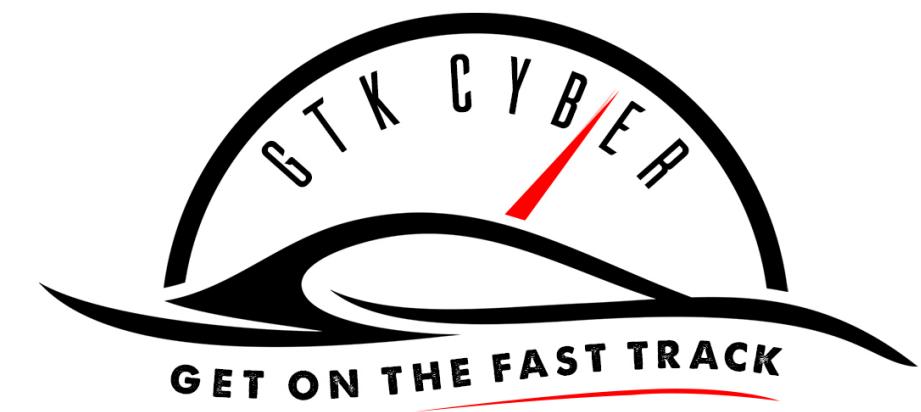
df_from_rdd.show()
df_from_rdd.printSchema()
```

ClientID	DateTime	Endpoint	Host	Method	Protocol	Response	SizeBytes	UserID
-	2012-08-03 17:09:47 -0700	/html/quickcrossf...	208.115.113.85	GET	HTTP/1.1	200	8770	-
-	2012-08-03 17:09:47 -0700	/geotechnical/pro...	208.115.113.85	GET	HTTP/1.1	200	20813	-
-	2012-08-03 17:09:47 -0700	/geotechnical/pro...	208.115.113.85	GET	HTTP/1.1	200	17070	-
100.92.69.23 -	2012-08-03 17:09:47 -0700	100.92.69.23 -	200	443	HTTP/1.1	200	40001	-

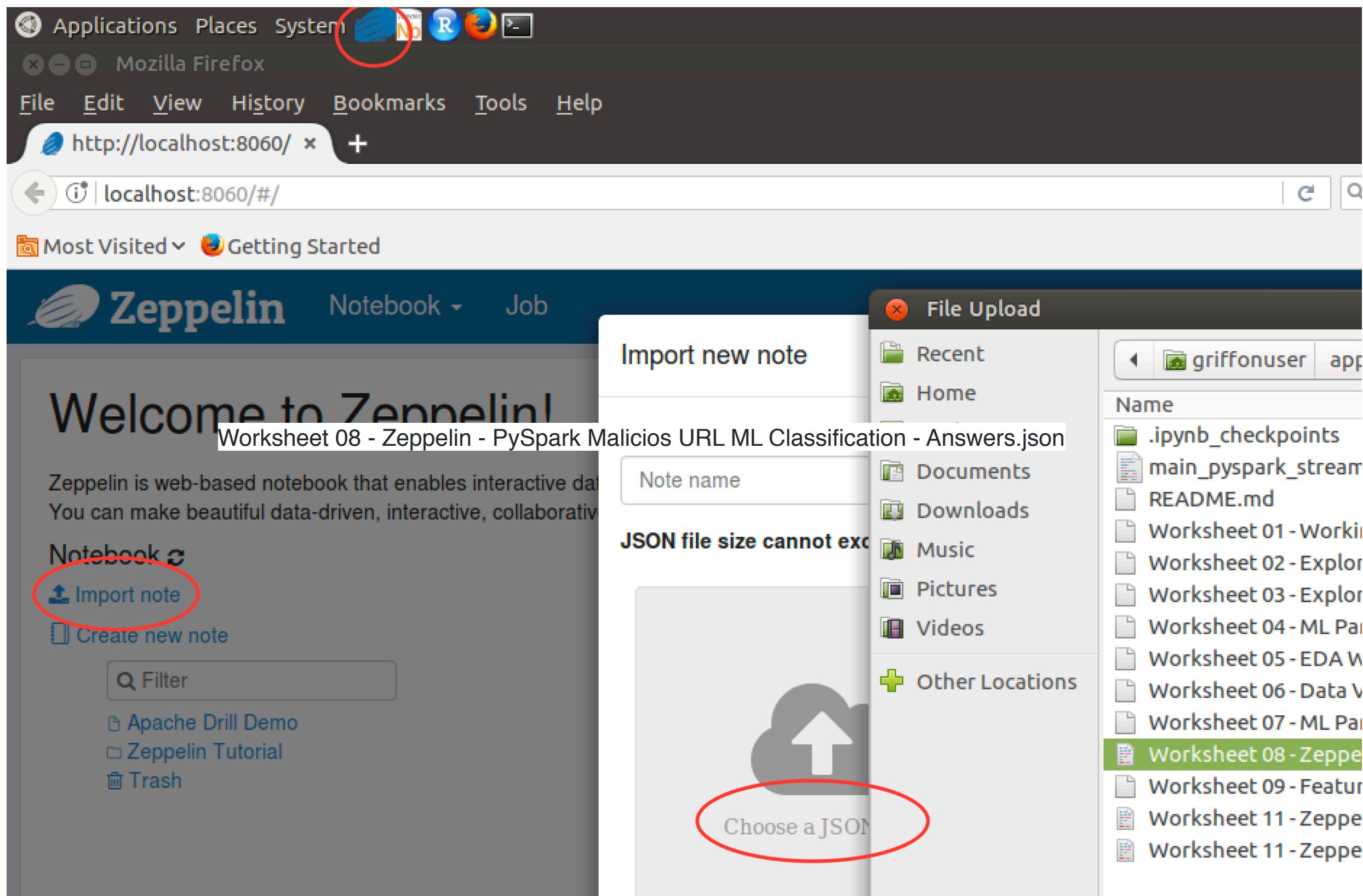


sqlContext - DataFrames: ETL

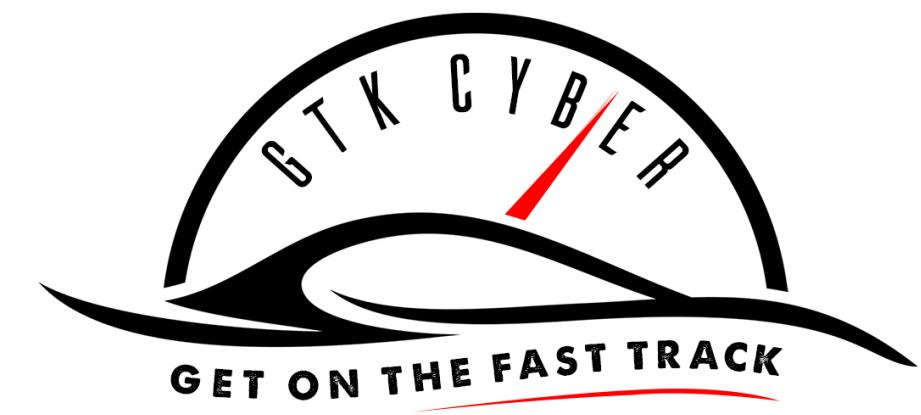
pyspark.ml - DataFrames: Machine Learning



ETL and Machine Learning: Detection of malicious URLs in PySpark

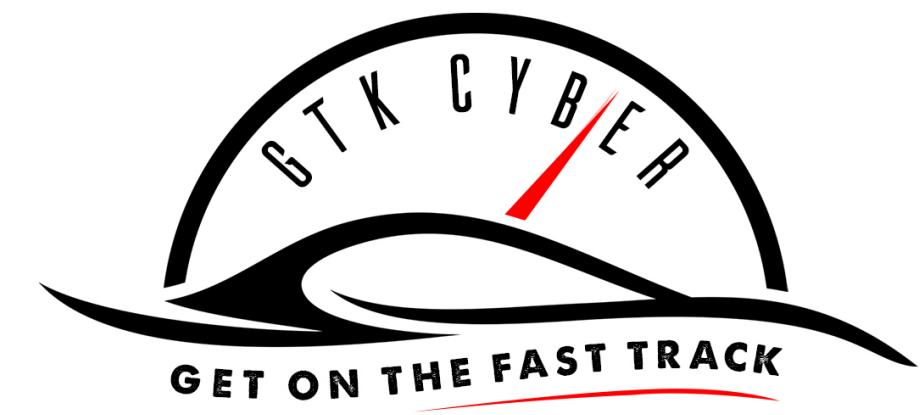


Bonus Worksheet 1 - Zeppelin - PySpark Malicious URL ML Classification - Answers.json



Import PySpark and Python libraries

```
%pyspark
# PySpark
from pyspark.sql.types import *
from pyspark.sql.functions import *
from pyspark.ml import feature
from pyspark.ml.classification import RandomForestClassifier, DecisionTreeClassifier
from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvaluator
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
# Python
import json
import pandas as pd
import numpy as np
import re, sys, os
from datetime import datetime
```



sc, sqlContext and spark

SparkContext represents the connection to a Spark cluster and is the main entry point for Spark functionality. In Zeppelin notebook **sc** as well as **sqlContext** and **spark** session are automatically loaded.

In a regular Python script you would load them as follows and run the Python script in a terminal with **spark-submit <path/to/script>.py**

```
from pyspark import SparkConf, SparkContext
from pyspark.sql import SQLContext
from pyspark.sql import SparkSession

## CREATE SparkContext sc
conf = (SparkConf()
        .setMaster("local")
        .setAppName("Test1")
        .set("spark.executor.memory", "6g"))
sc = SparkContext(conf = conf)

## CREATE SQLContext sqlContext
sqlContext = SQLContext(sc)

## CREATE SparkSession
spark = SparkSession(sc) \
        .builder \
        .appName("Test2") \
        .getOrCreate()
```



Load csv data using sqlContext

```
# PySpark version
customSchema = StructType([
    StructField("url", StringType(), True), \
    StructField("isMalicious", IntegerType(), True), \
    StructField("domain", StringType(), True), \
    StructField("created", TimestampType(), True), \
    StructField("isIP", StringType(), True), \
    ...
])
fname = DATA_HOME + 'url_data_small.csv'

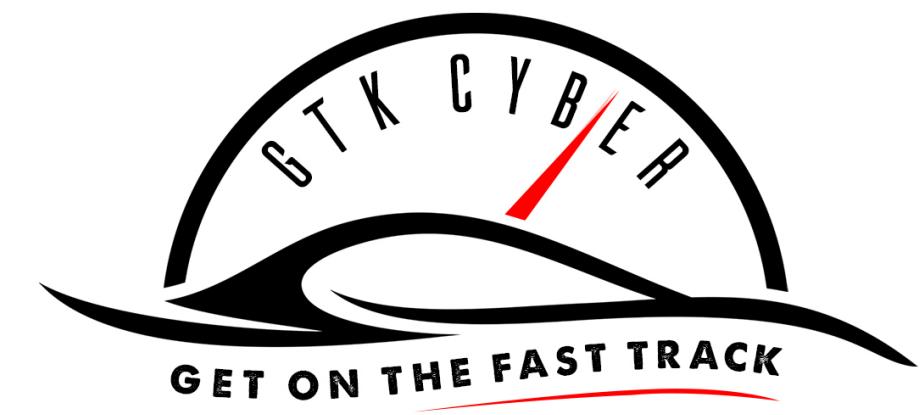
# Version 1: pre-defining schema
df = sqlContext.read \
    .format('com.databricks.spark.csv') \
    .options(header='true') \
    .load(fname, schema=customSchema)

# Version 2: Letting Spark infer Schema for you
df = sqlContext.read \
    .format('com.databricks.spark.csv') \
    .options(header='true', inferSchema="true") \
    .load(fname)

# Drop certain columns, easier to use select method
df = df.select([c for c in df.columns if c not in ['domain']])
print('Length of DataFrame: ', df.count())
df.show(5)
df.printSchema()
```

PySpark DataFrames are similar to pandas

<http://spark.apache.org/docs/latest/sql-programming-guide.html>



ML Feature Engineering: Apply custom Python function to df

```
%pyspark
def H_entropy(x):
    # Calculate Shannon Entropy
    prob = [ float(x.count(c)) / len(x) for c in dict.fromkeys(list(x)) ]
    H = - np.sum([ p * np.log2(p) for p in prob ])
    return np.float(H)

def digits(x):
    # Find Number of Digits
    pattern = re.compile('([0-9])')
    return len(re.findall(pattern, x))

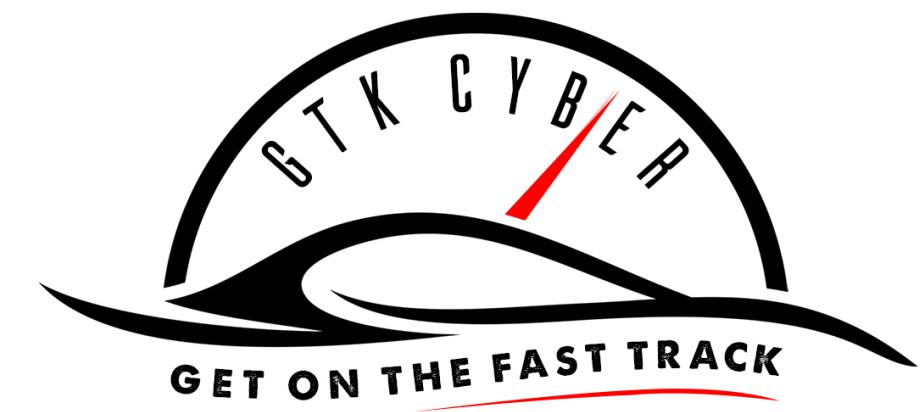
def calc_duration(x):
    # Calculate time difference
    delta = datetime.today() - x
    return delta.days
```

```
%pyspark
# Define User Defined Function
udf_length = udf(lambda x: len(x))
udf_entropy = udf(lambda x: H_entropy(x))
udf_digits = udf(lambda x: digits(x))
udf_duration = udf(lambda x: calc_duration(x))

# Apply UDF to one column of the df and add as new column to it.
df2 = df.withColumn('Length', udf_length(col('url')).cast(IntegerType()))\
         .withColumn('EntropyDomain', udf_entropy(col('url')).cast(FloatType()))\
         .withColumn('DigitsCount', udf_digits(col('url')).cast(IntegerType()))\
         .withColumn('DurationCreated', udf_duration(col('created')).cast(IntegerType()))
df2.show(5)
```

Spark User Defined Functions

<https://docs.databricks.com/spark/latest/spark-sql/udf-in-python.html>



ML example: Load feature matrix

```
fname = DATA_HOME + 'url_features_final_df_spark.csv'  
df = sqlContext.read \  
    .format('com.databricks.spark.csv') \  
    .options(header='true', inferSchema="true") \  
    .load(fname)  
print(df.columns)  
print('Length of DataFrame: ', df.count(), '\nColumns: ', len(df.columns))
```

```
['isMalicious', 'isIP', 'Length', 'LengthDomain', 'DigitsCount', 'EntropyDomai  
'us', 'cc', 'pt', 'tv', 'cokr', 'su', 'ws', 'orguk', 'club', 'pro26', 'all', 'n', 'new', 'news43', 'nte', 'ort', 'pro46', 'res', 'sta', 'ste', 'ter', 'the51  
mazon', 'amazonaws', 'ancestry', 'annstringer', 'answers', 'apple', 'aquatixbo  
, 'cndoubleegret', 'co', 'computantest', 'dedivan', 'directxex', 'downf468', 'gmart', 'home', 'iask', 'imdb', 'in', 'instagram', 'intnet', 'ip', 'jaaeza', 'manta', 'mhicable', 'mobile', 'muralove', 'myjino', 'mylife', 'myspace', 'myut  
ingshu8', 'pinterest', 'plus', 'qq', 'qualifiedplans', 'reia', 'rumormillnews', 'tripod', 'tvseriesfinale', 'twitter', 'u', 'unitedstatesreferral', 'usa', ube', 'DurationCreated']
```

Length of DataFrame: 87380

Columns: 158



Getting features and target vector in correct format

```
%pyspark
```

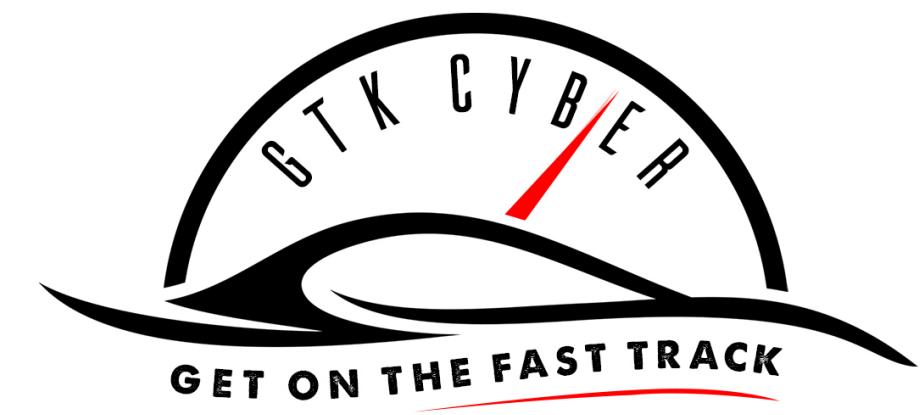
```
features_names = df.columns
features_names.remove('isMalicious')

assembler = feature.VectorAssembler(inputCols=features_names, outputCol='features')
data = assembler.transform(df).select('features', 'isMalicious') # X (features) and target(label)
data.show(5)
```

```
+-----+-----+
|      features|isMalicious|
+-----+-----+
|(157,[1,2,3,4,5,8...]|      0|
|(157,[1,2,4,6,156...]|      0|
|(157,[1,2,3,4,5,6...]|      0|
|(157,[1,2,4,6,50,...|      0|
|(157,[1,2,3,4,5,6...]|      0|
+-----+-----+
only showing top 5 rows
```

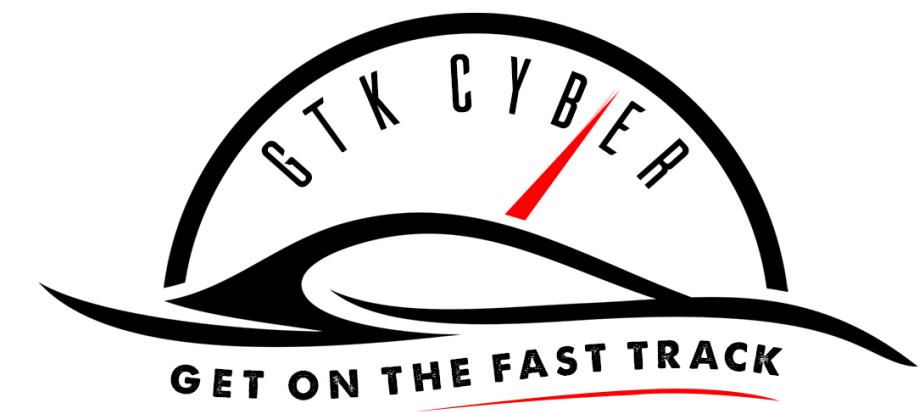
As opposed to Python's sklearn, in PySpark **features** and **target** labels will remain in **ONE** DataFrame!!!

- Column that contains the target labels has to be explicitly declared
- Features will be assembled to a vector or more specifically a SparseVector.



Simple train test split

```
%pyspark
## split train-test
seed=333
(train, test) = data.randomSplit([0.8,0.2], seed=seed)
train.show(5)
train.printSchema()
train.cache()
test.cache()
print(train.take(1))
```

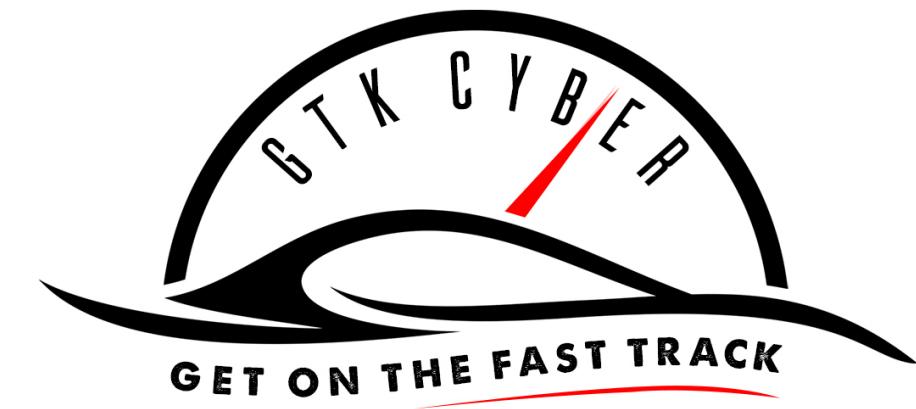


Spark ML Lib: Decision Tree Classifier

```
%pyspark
## train model
clf = DecisionTreeClassifier(featuresCol="features",labelCol="isMalicious")
# clf = RandomForestClassifier(featuresCol="features",labelCol="isMalicious")
# fit model
model = clf.fit(train)
# make predictions
predictions = model.transform(test)
predictions.show(5)
```

```
+-----+-----+-----+-----+
|      features|isMalicious| rawPrediction| probability|prediction|
+-----+-----+-----+-----+
|(157,[1,2,3,4,5,6...]|      0|[7239.0,1018.0]|[0.87671066973477...|    0.0|
|(157,[1,2,3,4,5,6...]|      0|[7693.0,250.0]|[0.96852574593982...|    0.0|
|(157,[1,2,3,4,5,6...]|      0|[7239.0,1018.0]|[0.87671066973477...|    0.0|
|(157,[1,2,3,4,5,6...]|      0|[7239.0,1018.0]|[0.87671066973477...|    0.0|
|(157,[1,2,3,4,5,6...]|      0|[4272.0,2722.0]|[0.61080926508435...|    0.0|
+-----+-----+-----+-----+
only showing top 5 rows
```

<http://spark.apache.org/docs/latest/api/python/pyspark.ml.html>

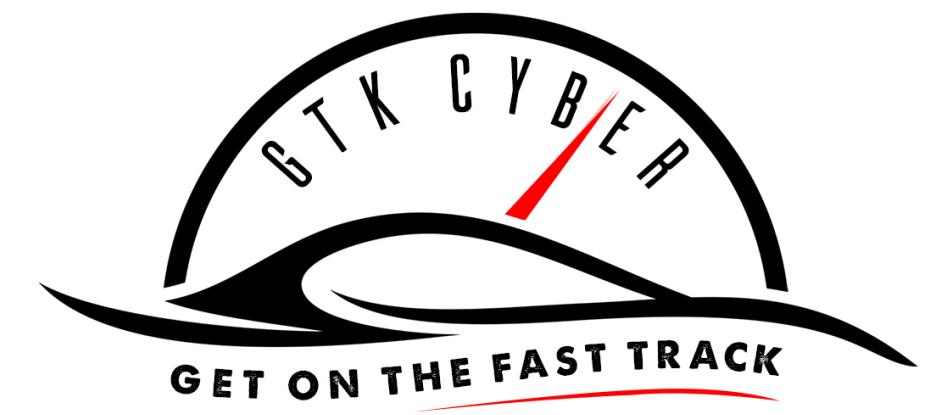


Model Evaluation

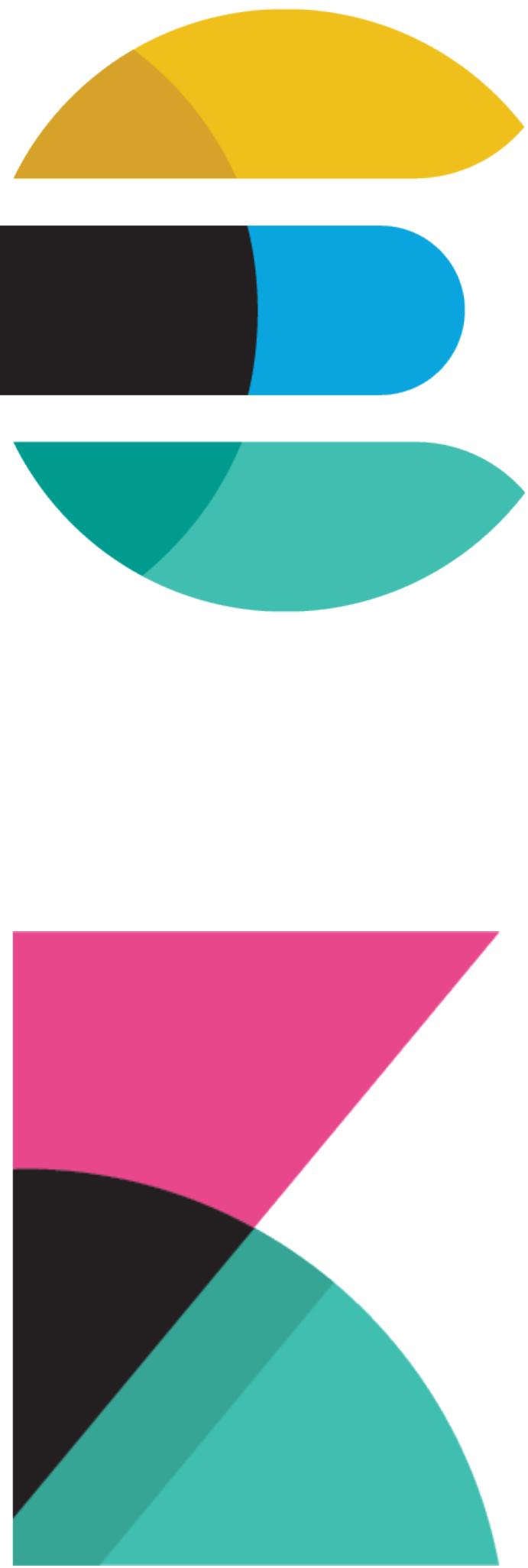
```
%pyspark
eval1 = BinaryClassificationEvaluator(rawPredictionCol="prediction", labelCol="isMalicious", metricName="areaUnderROC")
areaunderROC = eval1.evaluate(predictions)

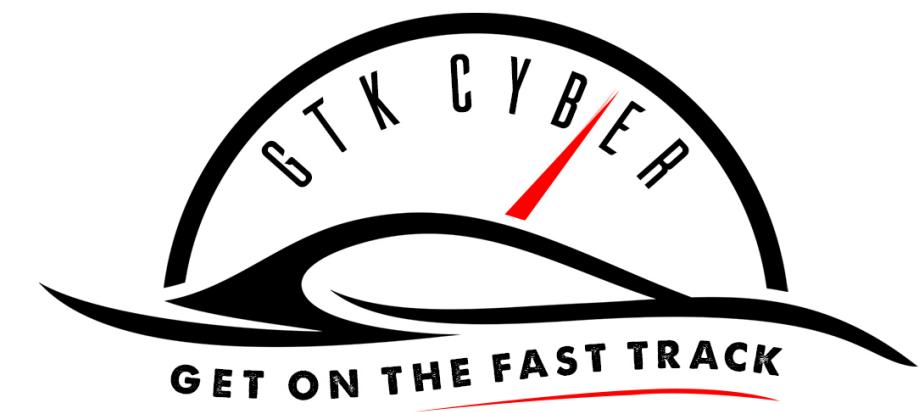
print('areaUnderROC of Decision Tree Classification: ', areaunderROC)
# print('Feature names: ', df.columns)
```

areaUnderROC of Decision Tree Classification: 0.8128223827076657



spark





Integrate Spark with ElasticSearch and Kibana



- ETL (Extract Transform Load)
- Machine Learning

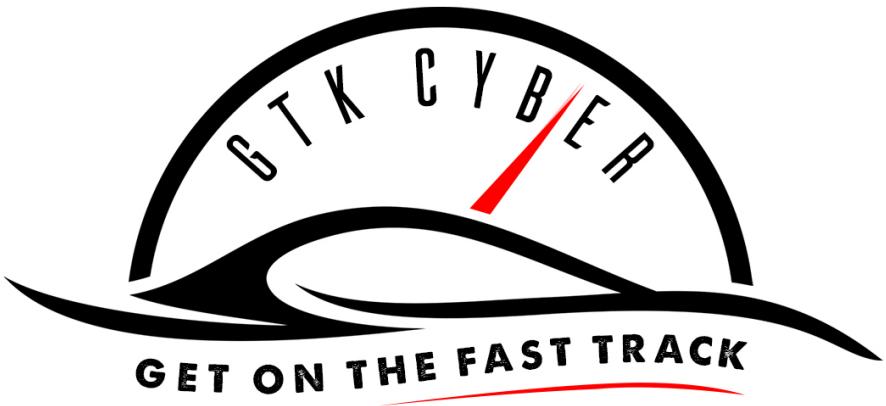


- Real-time indexing and searching
- Optimized for timestamped data



- Dashboards (geomap, barcharts, piecharts, timelion)

Bonus Worksheet 2 - Zeppelin Part1 - PySpark + ELK + Kafka - Answers.json



Batch Analysis of PCAP (csv) in PySpark

```
fname = DATA_HOME + 'maccdc2012_00000.csv.gz'

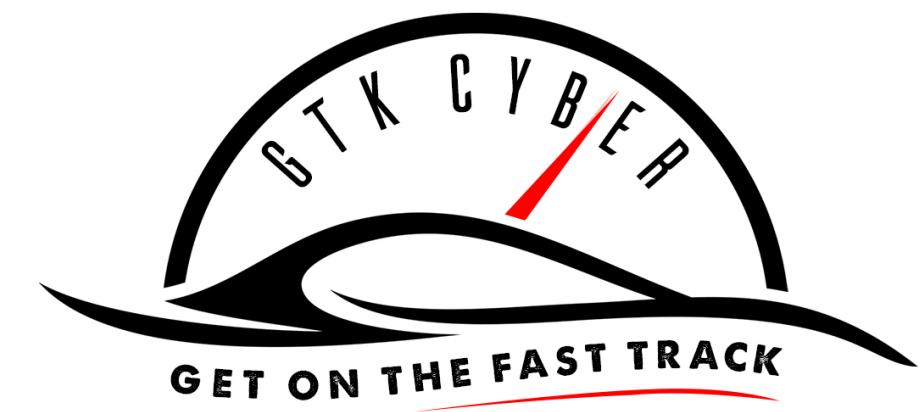
df = sqlContext.read\
    .format('com.databricks.spark.csv')\
    .options(delimiter="|", header="false", inferSchema="true")\
    .load(fname)

colNames = ["frameNumber", "frameTime", "frameProtocols", "frameLen", "tcpLen", "tcpHdrlen",
            "ipSrc", "ipDst", "tcpSrcport", "tcpDstport", "tcpFlags", "country", "city", "lat", "lon"]

mapping = dict(zip(['_c'+str(i) for i in range(len(df.columns))], colNames))
df = df.select([col(c).alias(mapping.get(c, c)) for c in df.columns])
df = df.na.drop(how='any')
# df.printSchema()

selectedCols = ["frameTime", "ipSrc", "ipDst", "tcpSrcport", "tcpDstport", "frameLen", "lat", "lon"]
df = df.select(selectedCols) # or select specific cols only
df.show()
df.printSchema()
```

frameTime	ipSrc	ipDst	tcpSrcport	tcpDstport	frameLen	lat	lon
Mar 16, 2012 08:3...	192.168.202.76	64.4.23.149	51655	40006	70	37.339401	-121.894997
Mar 16, 2012 08:3...	192.168.202.76	157.56.52.15	51658	40046	70	47.680099	-122.120598
Mar 16, 2012 08:3...	192.168.202.76	64.4.23.149	51659	80	70	37.339401	-121.894997
Mar 16, 2012 08:3...	192.168.202.76	157.55.56.150	51660	443	70	41.849998	-87.650002
Mar 16 2012 08:3...	192.168.202.76	157.55.56.150	51661	400301	70	53.3330991	-6.24891



Apply UDFs to DataFrame

```
%pyspark
def datetime_iso8601(x):

    date = datetime.strptime(str(x), '%b %d, %Y %H:%M:%S.%f000 %Z')
    date = date.isoformat()
    string = str(date)
    return string

def join_location(x, y):
    string = str(x) + "," + str(y)
    return string

toES_datetime = udf(lambda x: datetime_iso8601(x), StringType())
to_datetime = udf(lambda x: datetime.strptime(str(x), '%b %d, %Y %H:%M:%S.%f000 %Z'), TimestampType())
toLocation_str = udf(lambda x, y: join_location(x, y), StringType())

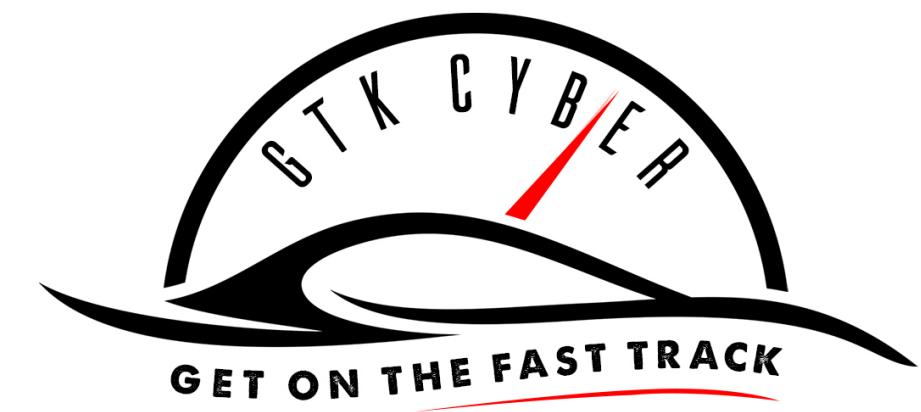
df = df.withColumn("@timestamp", toES_datetime(col("frameTime")))
        .withColumn("location", toLocation_str(col("lat"), col("lon")))
        .withColumn("frameTime", to_datetime(col("frameTime")))
df1 = df

df1 = df1.select(["ipSrc", "ipDst", "tcpSrcport", "tcpDstport", "@timestamp", "location", "frameLen"])

print(df1.count())
df1.cache() # save/cache, don't recompute
df1.show()
df1.printSchema()
```

```
2661
+-----+-----+-----+-----+-----+
|      ipSrc|      ipDst|tcpSrcport|tcpDstport| @timestamp| location|frameLen|
+-----+-----+-----+-----+-----+
|192.168.202.76| 64.4.23.149|      51655|     40006|2012-03-16T08:30:...|37.339401,-121.89...|      70|
|192.168.202.76| 157.56.52.15|      51658|     40046|2012-03-16T08:30:...|47.680099,-122.12...|      70|
```

- 1.) Transform datetime to isoformat and 'lat' and 'lon' to a format that geo_point in ES supports



Writing from PySpark to ElasticSearch

The screenshot shows the Kibana Dev Tools interface with the 'Console' tab selected. A command is being typed into the text area:

```
1 GET /_cat/indices?v
2 DELETE /bh-2017.07.08
3
4 GET _template/template_1
5 DELETE _template/template_1
6
7 PUT _template/template_1| ➤ ⌂
```

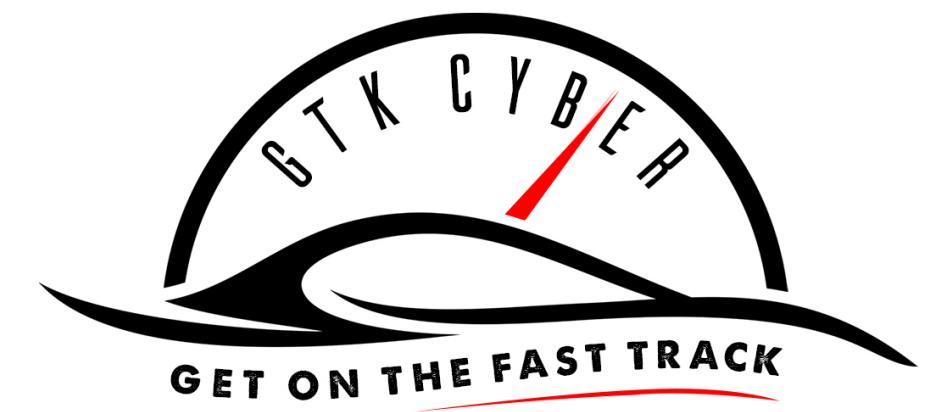
```
# Sink out to ElasticSearch
index_name = "bh-2017.07.08/pcap"

# Note: requires elasticsearch-hadoop jar corresponding
# to your ElasticSearch installation (https://jar-download.com/?search\_box=elasticsearch-hadoop)

df1.write.format("org.elasticsearch.spark.sql").mode("append")\
    .option("es.resource", index_name) \
    .option("es.index.auto.create", "true") \
    .option("es.mapping.date.rich", "true") \
    .option("es.nodes.wan.only", "false") \
    .option("es.net.ssl", "false") \
    .option("es.nodes", "localhost") \
    .option("es.port", "9200") \
    .save()
```

3.) Sink out to ES

2.) Define mapping in ES



Configure index pattern in ES

Screenshot of the Kibana Management / Kibana interface showing the configuration of an index pattern.

Left Sidebar: Management / Kibana

- Discover
- Visualize
- Dashboard
- Timelion
- Machine Learning
- Graph
- Dev Tools
- Monitoring
- Management

Top Bar: Management / Kibana

Warning: No default index pattern. You must select or create one to continue.

Configure an index pattern

In order to use Kibana you must configure at least one index pattern. Index patterns are used to id...

Index name or pattern: Patterns allow you to define dynamic index names using * as a wildcard. Example: logstash-*
bh-*

Index contains time-based events:

Time-field name: [refresh fields](#) @timestamp

Expand index pattern when searching: With this option selected, searches against any time-based index pattern that contains a wildcard will search across all documents in the index within the specified time range.

Searching against the index pattern `logstash-*` will actually query Elasticsearch for the specific matching index.

Use event times to create index names [DEPRECATED]:

Create

Right Panel: bh-*

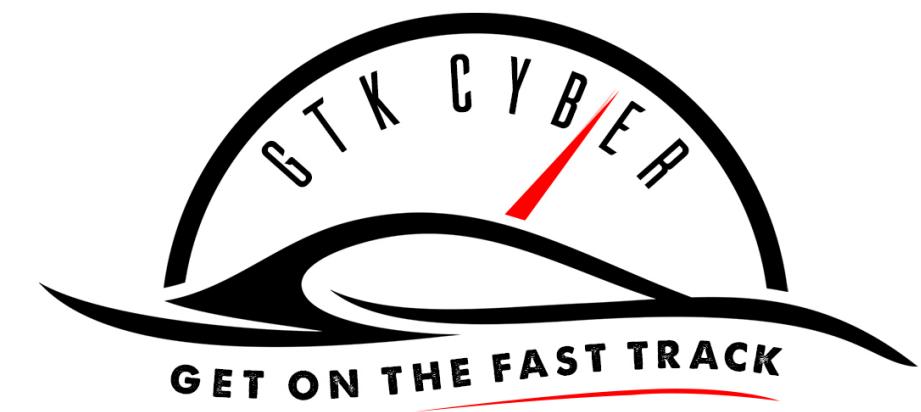
Configured time field: @timestamp

This page lists every field in the `bh-*` index and the field's associated core type as recorded by Elasticsearch. While this list allows you to view the core type of each field, changing field types must be done using Elasticsearch's [Mapping API](#).

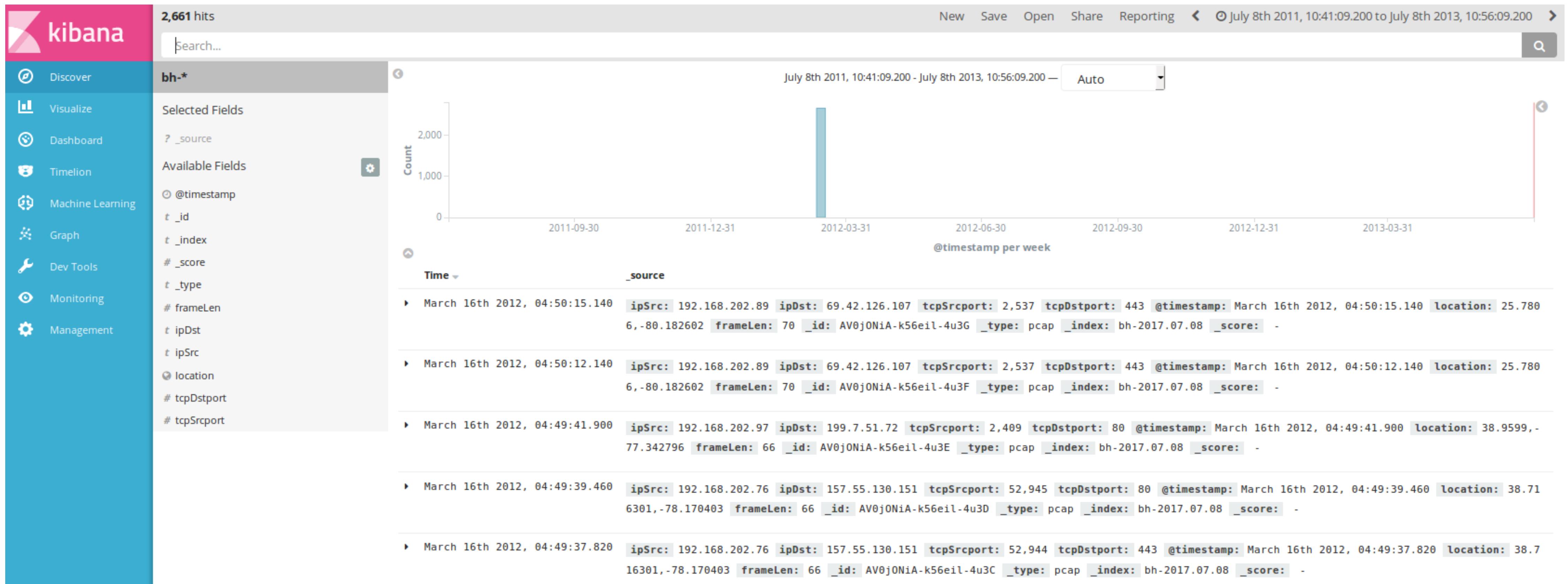
name	type	format	searchable	aggregatable	analyzed	excluded	controls
ipDst	string		✓			✓	
ipSrc	string		✓			✓	
tcpSrcport	number		✓	✓			
@timestamp	date		✓	✓			
_source	_source						
location	geo_point		✓	✓			
tcpDstport	number		✓	✓			
frameLen	number		✓	✓			
_id	string						
_type	string		✓	✓			
_index	string						
_score	number						

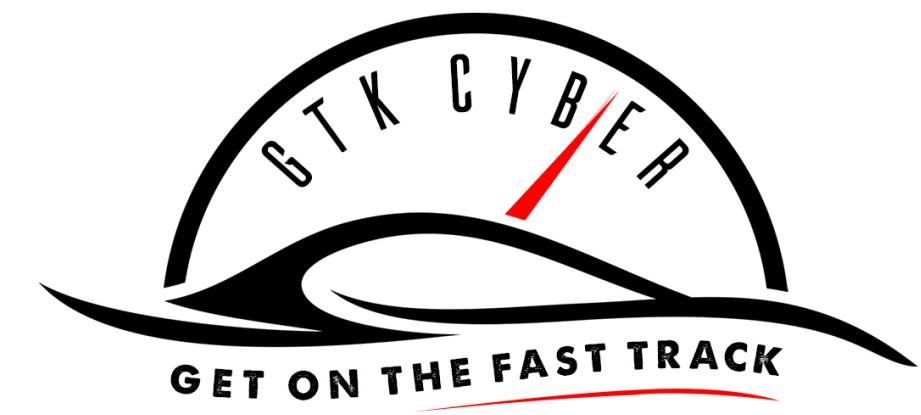
Scroll to top

Page Size 25

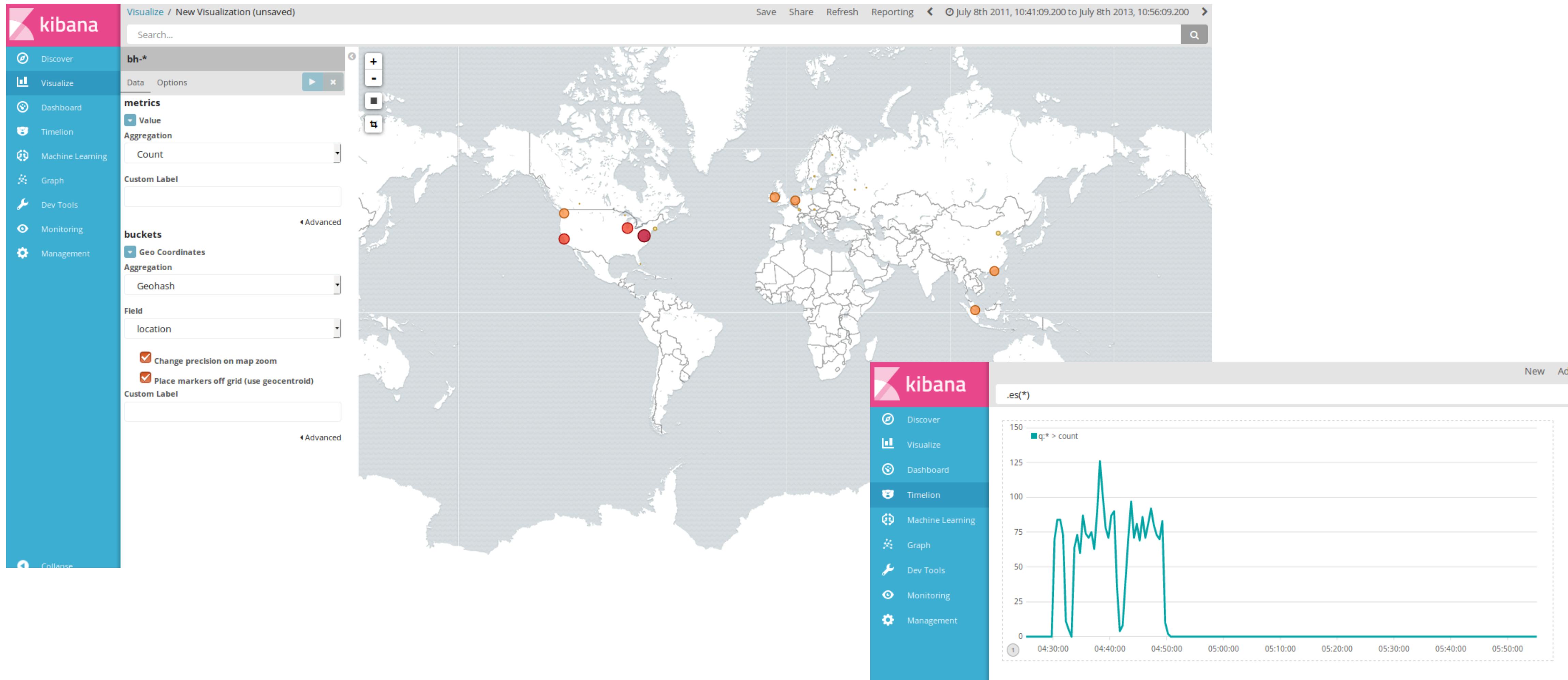


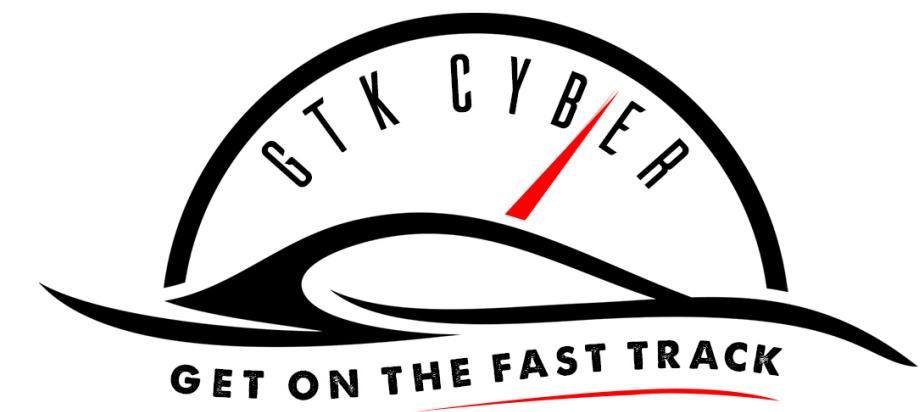
Searching in ES





Data Visualizations in Kibana





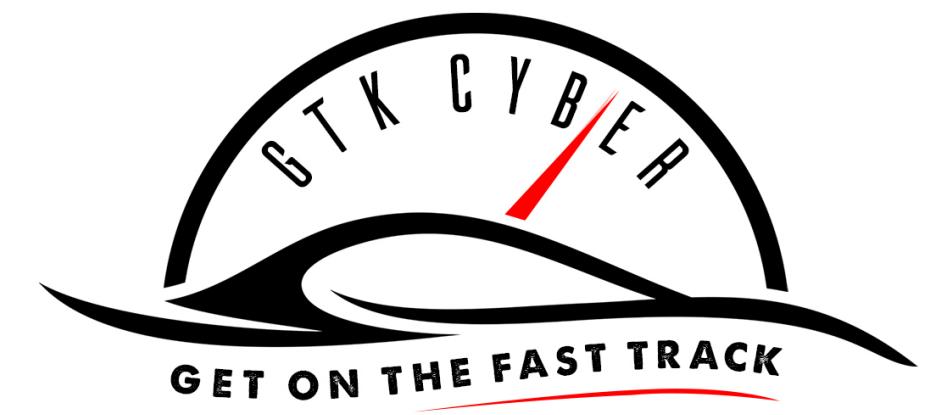
More ETL in PySpark (GroupBy and Aggregations)

```
%pyspark
exprs = [sum("frameLen").alias("frameLenSum"), count("ipSrc").alias("Npackets")]

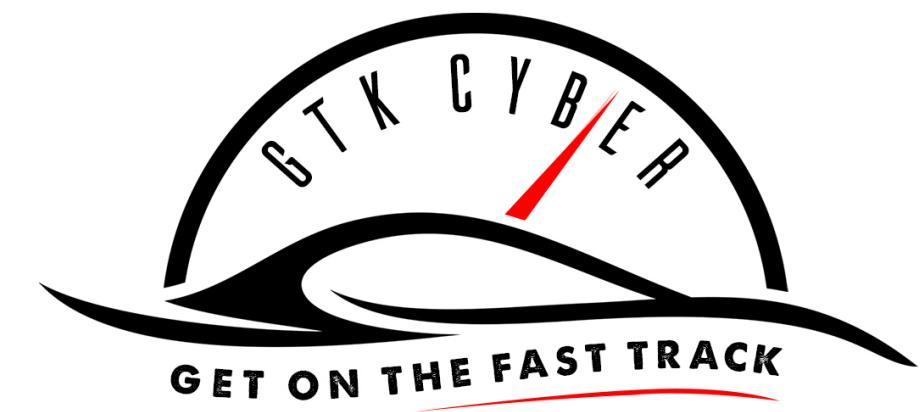
df_agg = df.groupBy([window("frameTime", "5 seconds"), "ipSrc", "ipDst"])\n    .agg(*exprs).orderBy("window.start").na.drop(how="any")

df_agg.show()
df_agg.count()
# df_agg.createOrReplaceTempView("features")

+-----+-----+-----+-----+-----+
|       window| ipSrc|      ipDst|frameLenSum|Npackets|
+-----+-----+-----+-----+-----+
|[2012-03-16 08:30...|192.168.202.76| 157.56.52.15|      350|      5|
|[2012-03-16 08:30...|192.168.202.76| 64.4.23.149|      280|      4|
```

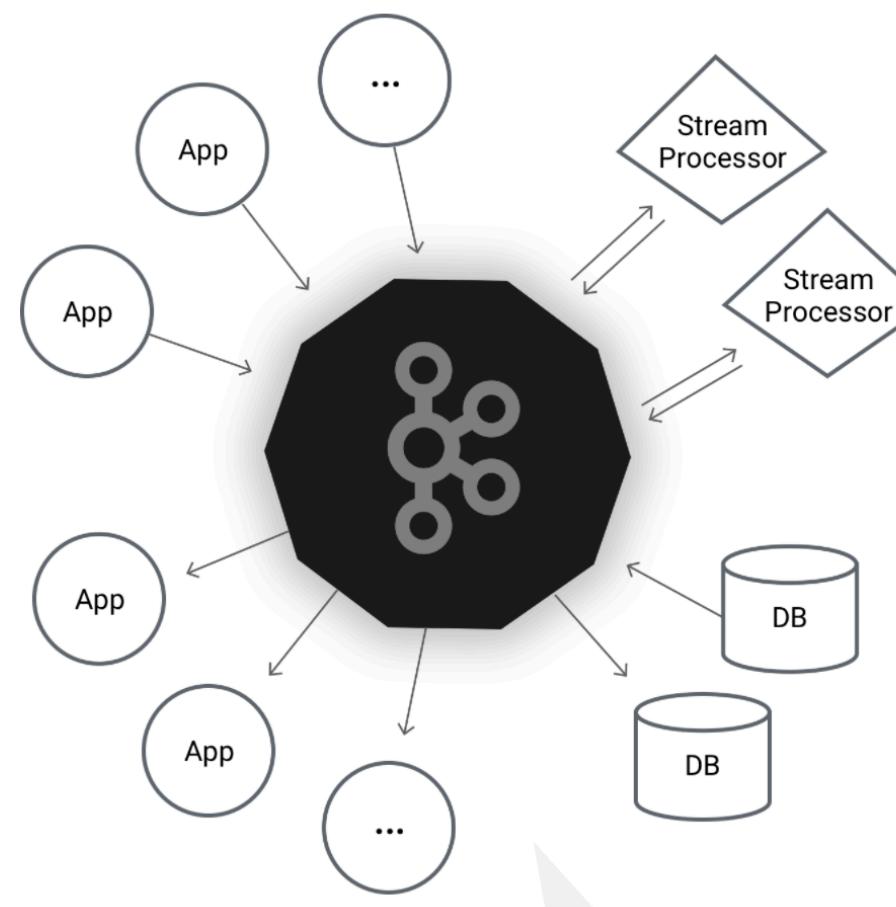


Spark

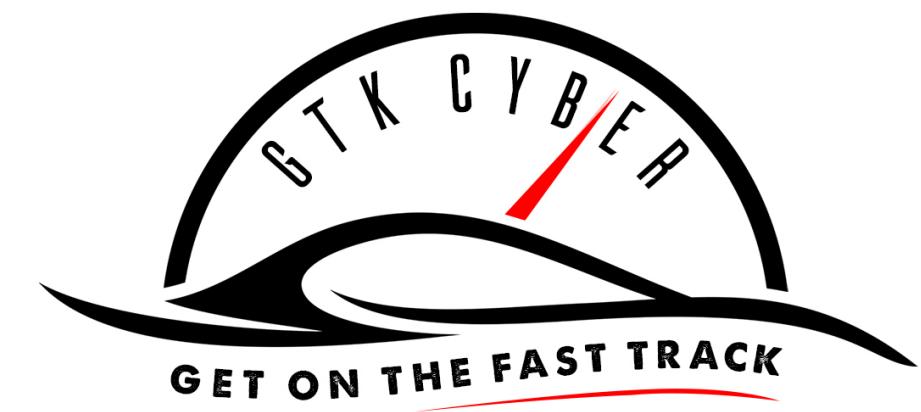


Kafka and Spark Structured Streaming

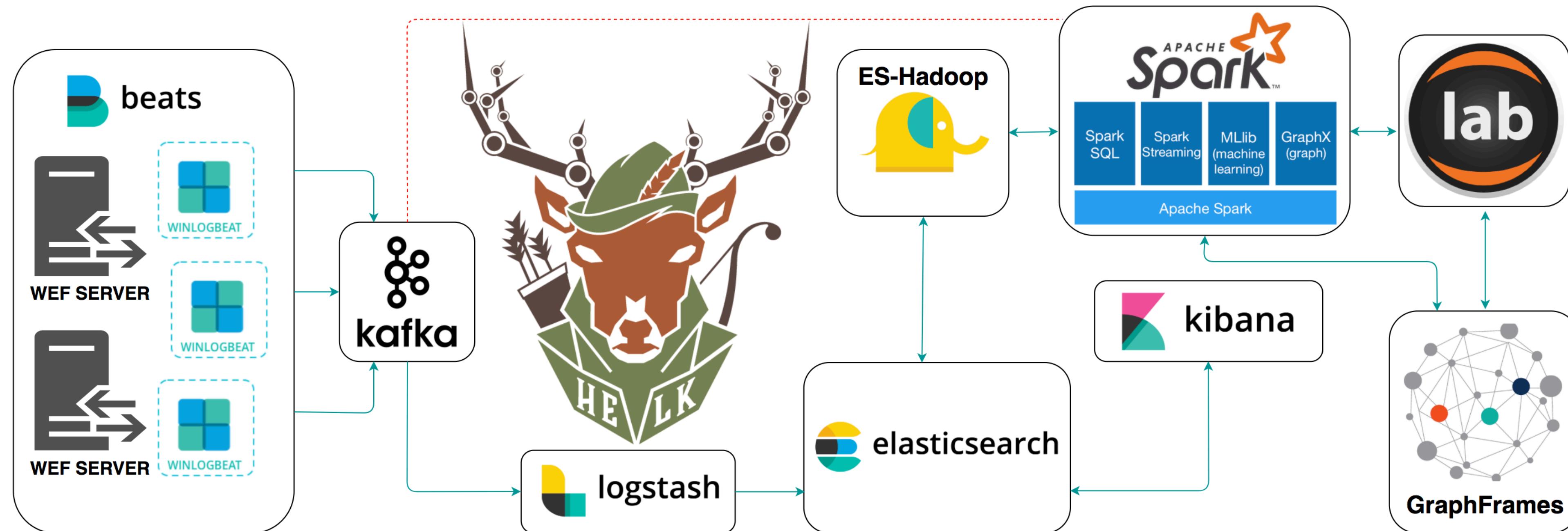
- PUBLISH & SUBSCRIBE to streams of data like a messaging system
- PROCESS streams of data efficiently and in real time
- STORE streams of data safely in a distributed replicated cluster



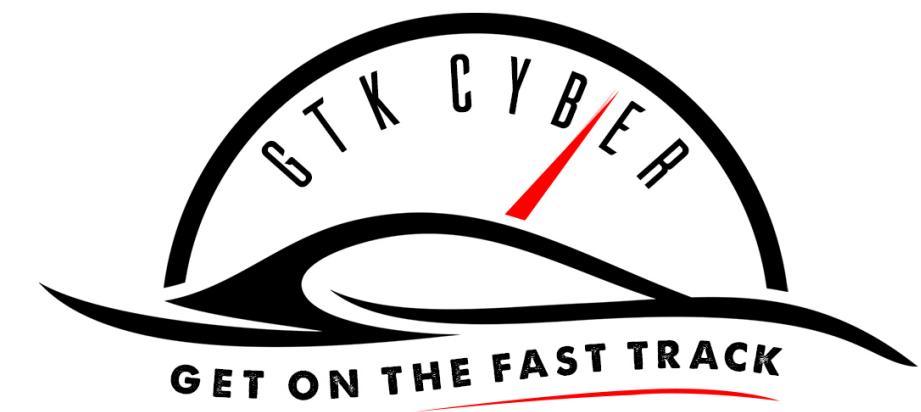
- Structured Streaming:
- Real-time analytics in micro-batches
 - sqlContext/DataFrames



A Hunting ELK (Elasticsearch, Logstash, Kibana)



<https://github.com/Cyb3rWard0g/HELK>



Create topic and initialize Kafka producer

```
# KAFKA
```

```
1. $KAFKA_HOME/bin/zookeeper-server-start.sh $KAFKA_HOME/config/zookeeper.properties (Start Zookeeper)
2. $KAFKA_HOME/bin/kafka-server-start.sh $KAFKA_HOME/config/server.properties (Start Kafka Server)
In GRIFFON skip step 1 and 2 and simply go to Applications > Big Data > Start Kafka
3. $KAFKA_HOME/bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic testlogs (Create example topic "testlogs")
cd applied-ds
4. $KAFKA_HOME/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic testlogs < data/stream/apache-access.log (Initialize Kafka Producer)
```

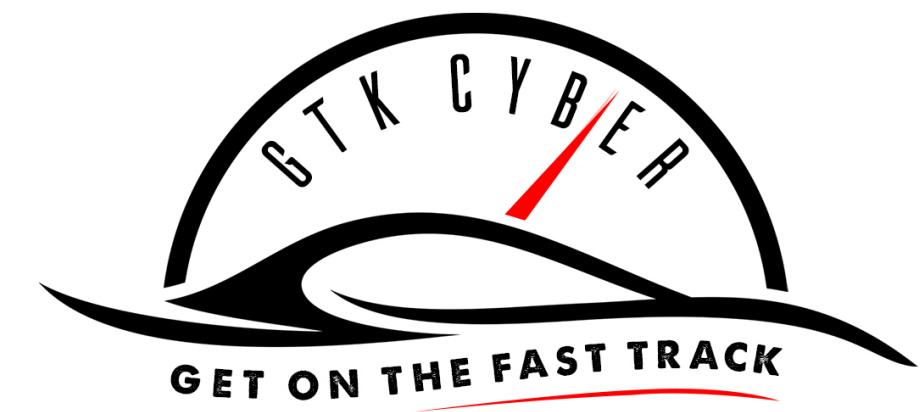
```
# Spark
```

```
5. Spark subscribe to Kafka topic using the Structured Streaming API (Kafka Consumer)
6. Spark DataFrame manipulations of incoming stream of messages
7. Spark Sink out (various options)
```

Docs:

- Kafka Quick Start (<https://kafka.apache.org/quickstart>)
- Spark Structured Streaming in micro-batches (<http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html>)
- Kafka Integration (<http://spark.apache.org/docs/latest/structured-streaming-kafka-integration.html>)
- Zeppelin interpreter settings (<https://zeppelin.apache.org/docs/0.5.0-incubating/interpreter/spark.html>)

Bonus Worksheet 3 - Zeppelin Part2 - PySpark + ELK + Kafka - Answers.json



Subscribe to Kafka topic in PySpark

```
%pyspark

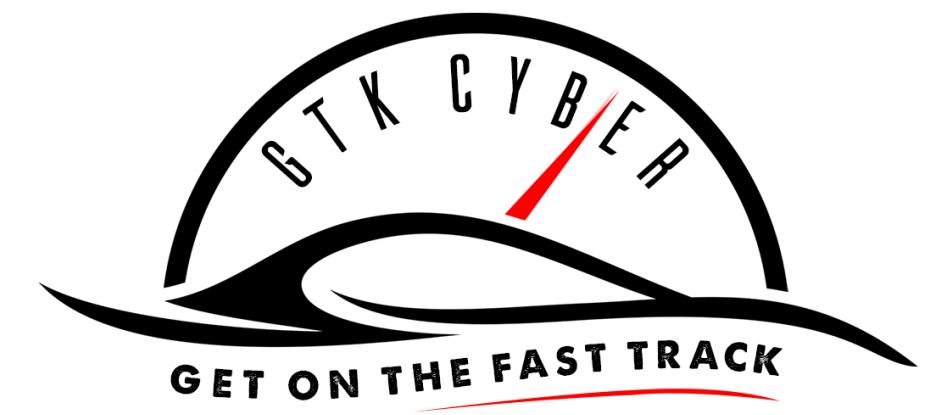
out_dir= DATA_HOME + "stream/out/out.csv"

while True:
    print("Kafka streaming running")
    ds0 = spark \
        .readStream \
        .format("kafka") \
        .option("kafka.bootstrap.servers", "localhost:9092") \
        .option("subscribe", "testlogs") \
        .load()

    # incoming values are in bytes, the next line deserializes the bytes and regular df operations can be applied
    ds0 = ds0.selectExpr("CAST(value AS STRING)", "CAST(timestamp AS STRING)")
    ds1 = parse_apache_log(ds0.select("value"))

    query = ds1.writeStream \
        .format("console") \
        .start()

    query.awaitTermination()
```

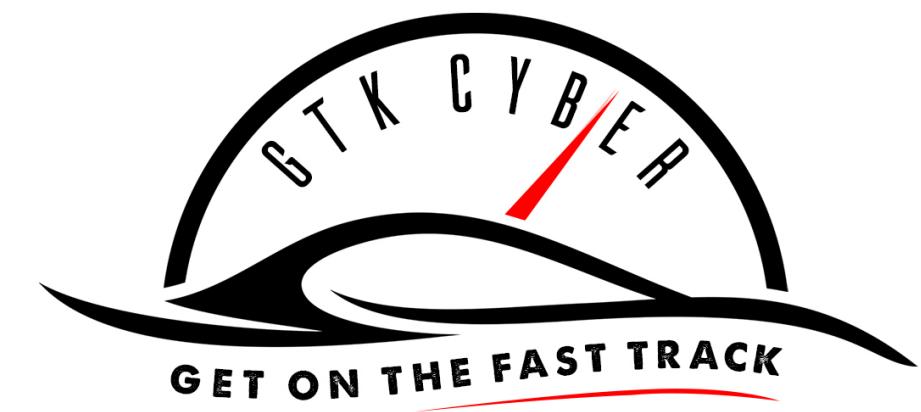


Parse Apache Access Log via DataFrame API

```
%pyspark
regex_pattern = '^(\\S+) (\\S+) (\\S+) \\[(\\w:/]+\\s[+\\-]\\d{4})\\] \"(\\S+) (\\S+) (\\S+)\" (\\d{3}) (\\d+)'
def parse_apache_log(df_raw):

    df = df_raw.select(regexp_extract('value', regex_pattern, 1).alias('Host'),
                        regexp_extract('value', regex_pattern, 2).alias('ClientID'),
                        regexp_extract('value', regex_pattern, 3).alias('UserID'),
                        regexp_extract('value', regex_pattern, 4).alias('DateTime'),
                        regexp_extract('value', regex_pattern, 5).alias('Method'),
                        regexp_extract('value', regex_pattern, 6).alias('Endpoint'),
                        regexp_extract('value', regex_pattern, 7).alias('Protocol'),
                        regexp_extract('value', regex_pattern, 8).alias('Response').cast(IntegerType()),
                        regexp_extract('value', regex_pattern, 9).alias('SizeBytes').cast(FloatType())))

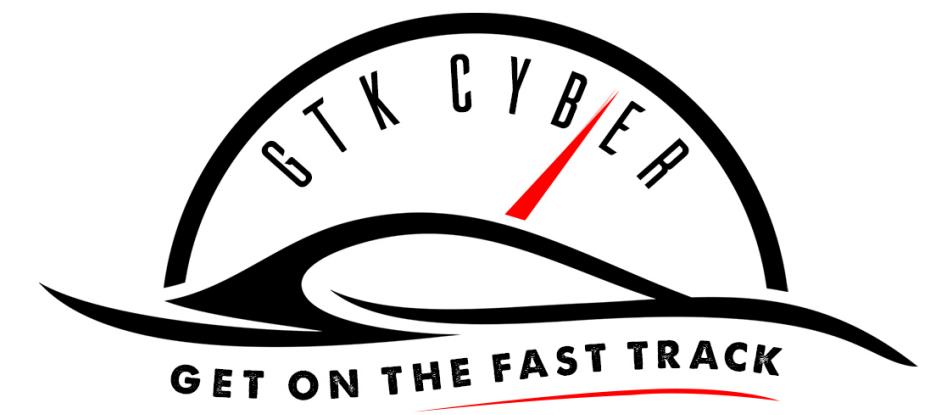
    df = df.select([c for c in df.columns if c not in ['ClientID', 'UserID', 'Method']])
    # print('Length of DataFrame: ', df.count())
    return df
```



Streaming Analytics Console Output

```
Kafka streaming running
-----
Batch: 0
-----
+---+-----+-----+-----+-----+
|Host|DateTime|Endpoint|Protocol|Response|SizeBytes|
+---+-----+-----+-----+-----+
+---+-----+-----+-----+-----+
-----  
Batch: 1
-----
+-----+-----+-----+-----+-----+-----+
|          Host|          DateTime|          Endpoint|Protocol|Response|SizeBytes|
+-----+-----+-----+-----+-----+-----+
| 208.115.113.85|03/Aug/2012:10:09...|/html/quickcrossf...|HTTP/1.1| 200| 8770.0|
| 208.115.113.85|03/Aug/2012:10:09...|/geotechnical/pro...|HTTP/1.1| 200| 20813.0|
| 208.115.113.85|03/Aug/2012:10:09...|/geotechnical/pro...|HTTP/1.1| 200| 17070.0|
| 208.115.113.85|03/Aug/2012:10:09...|/geotechnical/pro...|HTTP/1.1| 200| 16885.0|
| 208.115.113.85|03/Aug/2012:10:09...|/geotechnical/pro...|HTTP/1.1| 200| 16077.0|
| 65.52.110.146|03/Aug/2012:10:23...|/geotechnical/ind...|HTTP/1.1| 200| 3664.0|
|          |          |          |          | null| null|
| 213.186.119.134|03/Aug/2012:10:48...|/geotechnical/pro...|HTTP/1.1| 200| 4217.0|
| 66.249.73.3|03/Aug/2012:10:49...|/robots.txt|HTTP/1.1| 404| 229.0|
|          |          |          |          | null| null|
| 180.76.6.26|03/Aug/2012:11:00...|/geotechnical/acc...|HTTP/1.1| 302| 26.0|
```

main_pyspark_streaming_apachelogs_kafka.py



Questions?