Data Transformations

You won't always get data in a convienent format, often you will have to deal with data that is non-numerical, such as customer names, or zipcodes, country names, etc...

A big part of working with data is using your own domain knowledge to build an intuition of how to deal with the data, sometimes the best course of action is to drop the data, other times feature-engineering is a good way to go, or you could try to transform the data into something the Machine Learning Algorithms will understand.

Spark has several built in methods of dealing with thse transformations, check them all out here: http://spark.apache.org/docs/latest/ml-features.html (http://spark.apache.org/docs/latest/ml-features.html)

Let's see some examples of all of this!

In [2]:
```
from pyspark.sql import SparkSession
```

In [3]:
```
spark = SparkSession.builder.appName('data').getOrCreate()
```

In [4]:
```
df = spark.read.csv('fake_customers.csv',inferSchema=True,header=True)
```

In [5]:
```
df.show()
```

```
+-------+----------+-----+
|   Name|     Phone|Group|
+-------+----------+-----+
|   John|4085552424|    A|
|   Mike|3105552738|    B|
| Cassie|4085552424|    B|
|  Laura|3105552438|    B|
|  Sarah|4085551234|    A|
|  David|3105557463|    C|
|   Zach|4085553987|    C|
|  Kiera|3105552938|    A|
|  Alexa|4085559467|    C|
|Karissa|3105553475|    A|
+-------+----------+-----+
```

Data Features

StringIndexer

We often have to convert string information into numerical information as a categorical feature. This is easily done with the StringIndexer Method:

In [6]:
```
from pyspark.ml.feature import StringIndexer

df = spark.createDataFrame(
    [(0, "a"), (1, "b"), (2, "c"), (3, "a"), (4, "a"), (5, "c")],
    ["user_id", "category"])

indexer = StringIndexer(inputCol="category", outputCol="categoryIndex")
indexed = indexer.fit(df).transform(df)
indexed.show()
```

```
+-------+--------+-------------+
|user_id|category|categoryIndex|
+-------+--------+-------------+
|      0|       a|          0.0|
|      1|       b|          2.0|
|      2|       c|          1.0|
|      3|       a|          0.0|
|      4|       a|          0.0|
|      5|       c|          1.0|
+-------+--------+-------------+
```

The next step would be to encode these categories into "dummy" variables.

In [ ]:
```

```

VectorIndexer

VectorAssembler is a transformer that combines a given list of columns into a single vector column. It is useful for combining raw features and features generated by different feature transformers into a single feature vector, in order to train ML models like logistic regression and decision trees. VectorAssembler accepts the following input column types: all numeric types, boolean type, and vector type. In each row, the values of the input columns will be concatenated into a vector in the specified order.

Assume that we have a DataFrame with the columns id, hour, mobile, userFeatures, and clicked:

```
 id | hour | mobile | userFeatures     | clicked
----|------|--------|------------------|---------
 0  | 18   | 1.0    | [0.0, 10.0, 0.5] | 1.0
```

userFeatures is a vector column that contains three user features. We want to combine hour, mobile, and userFeatures into a single feature vector called features and use it to predict clicked or not. If we set VectorAssembler's input columns to hour, mobile, and userFeatures and output column to features, after transformation we should get the following DataFrame:

```
 id | hour | mobile | userFeatures     | clicked | features
----|------|--------|------------------|---------|-----------------------------
 0  | 18   | 1.0    | [0.0, 10.0, 0.5] | 1.0     | [18.0, 1.0, 0.0, 10.0, 0.5]
```

In [14]:
```
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler

dataset = spark.createDataFrame(
    [(0, 18, 1.0, Vectors.dense([0.0, 10.0, 0.5]), 1.0)],
    ["id", "hour", "mobile", "userFeatures", "clicked"])
dataset.show()
```

```
+---+----+------+--------------+-------+
| id|hour|mobile|  userFeatures|clicked|
+---+----+------+--------------+-------+
|  0|  18|   1.0|[0.0,10.0,0.5]|    1.0|
+---+----+------+--------------+-------+
```

In [15]:
```
assembler = VectorAssembler(
    inputCols=["hour", "mobile", "userFeatures"],
    outputCol="features")

output = assembler.transform(dataset)
print("Assembled columns 'hour', 'mobile', 'userFeatures' to vector column 'features'")
output.select("features", "clicked").show()
```

```
Assembled columns 'hour', 'mobile', 'userFeatures' to vector column 'features'
+--------------------+-------+
|            features|clicked|
+--------------------+-------+
|[18.0,1.0,0.0,10....|    1.0|
+--------------------+-------+
```

There ar emany more data transformations available, we will cover them once we encounter a need for them, for now these were the most important ones.