# Spark Project

**Haitian Jiang 19307110022**

## Abstract

This report contains the multi-dimensional analysis of demographic information, including age distribution, birth distribution, gender distribution, living address and migration of citizens on the leaked MERNIS database, using the mighty Spark framework. Problems are catogized into easy, normal and hard by difficulty. For easy and normal problems, I use RDD to manipulate the data and get the mining result by relatively simple functional programming. When it comes to hard problems, dataframe is used for the regression and classification model. Simple models like Naive Bayes and Linear Regression perform well on these tasks.

## 1 Introduction

The data used in this project is the leaked MERNIS database on 2016. MERNIS is the "The Central Civil Registration System" in which names given to babies by their parents are registered. Most Turkish first name contains gender information, like Ouz for males and Tuçe for females. However, Turkish had not haven surnames until the Surname Law is enacted in 1934. According to the law, surnames can only indicate the stemma, but no gender information.[1] This pattern is also shown in the results.

## 2 Prepare the data

```
# set environment variables for runtime
import os
os.environ["JAVA_HOME"]="/opt/homebrew/Cellar/openjdk@11/11.0.12/"
os.environ["PYSPARK_PYTHON"]="/opt/miniconda3/envs/spark/bin/python"
os.environ["PYSPARK_DRIVER_PYTHON"]="/opt/miniconda3/envs/spark/bin/python"

from pyspark import SparkContext, SparkConf
from pyspark.ml.stat import ChiSquareTest
from pyspark.ml.linalg import Vectors
from pyspark.sql import SparkSession

conf = SparkConf().setAppName("lala").setMaster("local")
sc = SparkContext(conf=conf)

data_path = '../mernis/data_dump.sql'
dataset = sc.textFile(data_path)
dataset = dataset.map(lambda x: x.split('\t'))

# frequently used functions
from operator import add  # a more elegant way
is_male = lambda row: row[6] == "E"
is_female = lambda row: row[6] == "K"
```

[1] https://en.wikipedia.org/wiki/Turkish_name

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

# 3 Easy Problems

## 3.1 E1. 统计土耳其所有公民中年龄最大的男性

First I calculated the age of each person in the data set. I considered the ages at the end of year 2009, according to the description of the data set. Then I add the age information into the data set for convenience in further mining.

```python
# add last column as age
dataset = dataset.map(lambda row: row+[2009-int(row[8].split('/')[2])+1])

eldest_man = dataset.filter(is_male)\
            .sortBy(lambda row: int(row[8].split('/')[2]))\
            .filter(lambda row: row[17] < 150).take(1)

print(f"The eldest man is {eldest_man[0][2]} {eldest_man[0][3]} if alive.")
```

The eldest man is HASAN GEZER if alive.
Here I filtered out those who aged more than 150 because it's physiologically impossible. If not so, the eldest man would be ridiculously old.

## 3.2 E2. 统计所有姓名中最常出现的字母

This task follows the epitome of word count task for map reduce. Just take the standard way of mapping to (key, 1).

```python
# turn each row into a list of (char, 1)
char_ize = lambda row: list(map(lambda c:(c,1), row[2]+row[3]))
most_freq = dataset.flatMap(char_ize).reduceByKey(add).sortBy(lambda x:-x[1]).take(1)
print(f"The most frequent char is {most_freq[0][0]}, with frequency {most_freq[0][1]}.")
```

The most frequent char is A, with frequency 82319942.

## 3.3 E3. 统计该国人口的年龄分布，年龄段分（0-18、19-28、29-38、39-48、49-59、>60）

In this problem, I map the age into the index of the age group by comparing it to the upper bound, and then reduce it to count the frequency.

```python
def age_bin(row):
    upper_bound = [18, 28, 38 ,48, 59, float("inf")]
    for i, bnd in enumerate(upper_bound):
        if row[17] <= bnd:
            return (i, 1)
dataset.map(age_bin).reduceByKey(add).sortBy(lambda x:x[0]).collect()
```

| Age | Frequency |
|---|---|
| 0∼18 | not shown up |
| 19∼28 | 11517813 |
| 29∼38 | 12239447 |
| 39∼48 | 9689514 |
| 49∼59 | 7832022 |
| ⩾60 | 8332913 |

表 1: Age distribution

From the result we can see that the data set does not include information about citizens under 18 years old, which agrees to the description of the data set in the problem set. Also, the frequency of each age bin decreases as the age goes up, which is typical for an age distribution.

### 3.4 E4. 分别统计该国的男女人数，并计算男女比例

Sex ratio is defined as the number of male divided by the number of female. Naturally, the number would be greater than 1.

```
male = dataset.filter(is_male).count()
female = dataset.filter(is_female).count()
sex_ratio = male / female
print(f"Male: {male}; female: {female}")
```

Male: 24534483; female: 25077226
The sex ratio is 0.978.

### 3.5 E5. 统计该国男性出生率最高的月份和女性出生率最高的月份

```
male_month = dataset.filter(is_male)\
             .map(lambda row: (row[8].split("/")[1], 1))\
             .reduceByKey(add).collect()
female_month = dataset.filter(is_female)\
               .map(lambda row: (row[8].split("/")[1], 1))\
               .reduceByKey(add).collect()
```

| Month | Male | Female |
|---|---|---|
| 1 | 3911691 | 3913285 |
| 2 | 2315662 | 2373331 |
| 3 | 2525290 | 2599692 |
| 4 | 2071026 | 2113036 |
| 5 | 2088048 | 2129273 |
| 6 | 1712476 | 1750905 |
| 7 | 2050057 | 2199096 |
| 8 | 1573404 | 1597236 |
| 9 | 1715532 | 1735173 |
| 10 | 1698787 | 1735641 |
| 11 | 1447936 | 1464149 |
| 12 | 1398445 | 1430474 |
| other | 26129 | 35934 |

表 2: Birth month distribution

For both male and female, the month where most people born is January.
There are some outliers in the data set, of which the month is 0 or empty. I just catagorize all of them to other. For the data, January has a anomalous higher birth rate. This may due to the lack of original date of birth information so that some aged people just pick January 1st as their birthday to inform the registration system. This phenomenon can be seen in later problem.

### 3.6 E6. 统计哪个街道居住人口最多

I use the city, district and neighborhood information to locate one street in case there are streets of same name in different regions. In fact there indeed exists same name of street in different cities, as the same case in China.

```
street = dataset.map(lambda x:(f"{x[11]}/{x[12]}/{x[13]}/{x[14]}", 1))\
         .reduceByKey(add).sortBy(key=lambda x:-x[1]).take(1)
```

The street containing most citizens is YUNUS EMRE CADDESI street, in ESENLER BASAKSEHIR MAH, BASAKSEHIR district, ISTANBUL, with 10307 people living there.

### 3.7 E7. 找出 60 岁以上人口最多以及占比最多城市

```
plus_60 = dataset.filter(lambda x:x[17]>=60).map(lambda x:(x[11],1))\
          .reduceByKey(add).sortBy(lambda x:-x[1]).collect
all_pop = dataset.map(lambda x:(x[11],1)).reduceByKey(add).collect()
d = dict(plus_60)
old_prop = [(i[0], d[i[0]]/i[1]) for i in all_pop]
old_prop.sort(key=lambda x:-x[1])
```

The city with most citizen over 60 is ISTANBUL, there are 1166039 of these. The reason is that Istanbul is the largest city in Turkey and it itself has the most population.
The city with highest proportion of citizens over 60 is SINOP, the rate is 30.32%.

### 3.8 E8, N1. 分别统计男性和女性中最常见的 10 个姓

```
male_last = dataset.filter(is_male)\
            .map(lambda x: (x[3], 1)).reduceByKey(add)\
            .sortBy(lambda x:-x[1]).take(10)
female_last = dataset.filter(is_female)\
              .map(lambda x: (x[3], 1)).reduceByKey(add)\
              .sortBy(lambda x:-x[1]).take(10)
```

| Order | Surname | Male | Female |
|---|---|---|---|
| 1 | YILMAZ | 352338 | 355954 |
| 2 | KAYA | 244272 | 244100 |
| 3 | DEMIR | 231289 | 230428 |
| 4 | SAHIN | 201958 | 202155 |
| 5 | CELIK | 199622 | 199330 |
| 6 | YILDIZ | 195162 | 194060 |
| 7 | YILDIRIM | 191966 | 192835 |
| 8 | OZTURK | 178610 | 180292 |
| 9 | AYDIN | 177894 | 178501 |
| 10 | OZDEMIR | 164085 | 165924 |

表 3: Most popular surnames

The top 10 most common surnames are exactly the same for male and female, which in accord with what I found in the Turkish law in the introduction part. This fact also indicates that it is hard for us to connect the information of a person with his/her surname.

### 3.9 E9. 分别找出男女性占比差异最大和最小的城市

In this problem, I use the living address to locate the city of each citizen, otherwise the information is invalid.

```
male_city = dataset.filter(is_male).map(lambda x:(x[11],1))\
            .reduceByKey(add).collect()
female_city = dataset.filter(is_female).map(lambda x:(x[11],1))\
              .reduceByKey(add).collect()

male_city_d = dict(male_city)
female_city_d = dict(female_city)
sex_ratio = [(city, male_city_d[city]/female_city_d[city])
                    for city in male_city_d]
sex_ratio.sort(key=lambda x:x[1])
print(sex_ratio[0], sex_ratio[-1])
```

The city with lowest sex ratio is ELAZIG, and its sex ratio is 0.934.
The city with highest sex ratio is TUNCELI, and its sex ratio is 1.047.

### 3.10 E10. 找到同年同月同日生的人数最多的日期

```
dataset.map(lambda row: (row[8],1)).reduceByKey(add)\
.sortBy(lambda x:-x[1]).take(1)
```

The day with most people born is 1/1/1966. There are 180577 people born on that day. When dig further, we can see that the first ten day with most people born are all Jan. 1st, and the year varies from 1950s to 1970s. This pattern supports the conjecture in E5 and explains why January has such a high birth rate.

## 4 Normal Problems

### 4.1 N1. 分别统计男性和女性中最常见的 10 个姓

See E8

### 4.2 N2. 统计每个城市市民的平均年龄，统计分析每个城市的人又老龄化程度，判断当前城市是否处于老龄化社会

First see the average age of the cities. Python uses a complex object to store integers, so there is no concern for overflowing.

```
total_age = dataset.map(lambda x: (x[11], x[17])).reduceByKey(add).collect()
population = dataset.map(lambda x:(x[11], 1)).reduceByKey(add).collect()

age = dict(total_age)
pop = dict(population)
for city in age:
    age[city] /= pop[city]
avg_age = list(age.items())
avg_age.sort()
avg_age[:10]
```

The average ages of some cities list as follows.

| City | Average age |
|---|---|
| ADANA | 41.40 |
| ADIYAMAN | 40.26 |
| AFYONKARAHISAR | 43.85 |
| AGRI | 38.10 |
| AKSARAY | 41.78 |
| AMASYA | 45.73 |
| ANKARA | 42.45 |
| ANTALYA | 42.33 |
| ARDAHAN | 43.97 |
| ARTVIN | 46.65 |

表 4: Average ages

Then let's take a deeper look into the aging society.

```
age_60 = dataset.filter(lambda x:x[17]>=60).map(lambda x:(x[11],1))\
         .reduceByKey(add).collect()
age_65 = dataset.filter(lambda x:x[17]>=65).map(lambda x:(x[11],1))\
         .reduceByKey(add).collect()
over_60, over_65 = dict(age_60), dict(age_65)
for city in pop:
    over_60[city] /= pop[city]
```

```
    over_65[city] /= pop[city]
aging = {city: (over_60[city], over_65[city], over_60[city]>0.1 or over_65[city]>0.07)
         for city in pop}
list(aging.items())[:10]
```

Below is a table showing if these randomly picked cities are aging cities. Aging is indeed a problem for many cities in Turkey.

| City | 60+ | 65+ | Aging |
|---|---|---|---|
| BALIKESIR | 23.81% | 17.19% | True |
| TEKIRDAG | 16.27% | 11.22% | True |
| GUMUSHANE | 22.61% | 17.35% | True |
| BOLU | 23.43% | 17.26% | True |
| MALATYA | 17.87% | 12.63% | True |
| RIZE | 19.94% | 14.68% | True |
| AYDIN | 21.37% | 15.29% | True |
| CANAKKALE | 24.72% | 17.93% | True |
| ESKISEHIR | 19.59% | 13.76% | True |
| IGDIR | 13.76% | 9.52% | True |

表 5: Aging Cities

### 4.3 N3, N5. 计算一下该国前 10 大人口城市中，每个城市的人口生日最集中分布的是哪 2 个月

```
# get the 10 largest cities
population.sort(key=lambda x:-x[1])
ten_largest = [i[0] for i in population[:10]]

birth_month = dataset.filter(lambda x:x[11] in ten_largest)\
.map(lambda x: ((x[11], x[8].split("/")[1]),1)).reduceByKey(add).collect()

from collections import defaultdict
city_month = defaultdict(list)
for m in birth_month:
    city_month[m[0][0]].append((m[0][1], m[1]))
for c in city_month:
    city_month[c].sort(key=lambda x:-x[1])
    city_month[c] = city_month[c][:2]
```

From the result we can see that in all 10 largest cities, the top 2 months with most birth are all January and March. This alligns with the global result.

| City | Month | Frequency | Month | Frequency |
|---|---|---|---|---|
| ISTANBUL | Jan | 1230125 | Mar | 883866 |
| MERSIN | Jan | 189829 | Mar | 110903 |
| KOCAELI | Jan | 138963 | Mar | 104654 |
| IZMIR | Jan | 383898 | Mar | 281508 |
| ANKARA | Jan | 451692 | Mar | 318455 |
| ANTALYA | Jan | 207555 | Mar | 134923 |
| BURSA | Jan | 245096 | Mar | 177899 |
| ADANA | Jan | 275762 | Mar | 134684 |
| KONYA | Jan | 204352 | Mar | 138089 |
| AYDIN | Jan | 193720 | Mar | 143582 |

表 6: Top 2 month with most births

### 4.4 N4. 统计该国前 10 大人口城市中，每个城市的前 3 大姓氏，并分析姓氏与所在城市是否具有相关性

Use the information of 10 most populated cities from last problem.

```
from collections import defaultdict

last_name = dataset.filter(lambda x:x[11] in ten_largest)\
            .map(lambda x: ((x[11], x[3]),1)).reduceByKey(add).collect()
city_lname = defaultdict(list)

for m in last_name:
    city_lname[m[0][0]].append((m[0][1], m[1]))

for c in city_lname:
    city_lname[c].sort(key=lambda x:-x[1])
    city_lname[c] = city_lname[c][:3]
print(city_lname)
```

| City | Surname | Freq | Surname | Freq | Surname | Freq |
|---|---|---|---|---|---|---|
| ADANA | YILMAZ | 16223 | KAYA | 13187 | DEMIR | 11550 |
| ANTALYA | YILMAZ | 21057 | KAYA | 12566 | CELIK | 12092 |
| MERSIN | YILMAZ | 15786 | SAHIN | 11593 | KAYA | 10356 |
| ISTANBUL | YILMAZ | 139142 | KAYA | 87341 | DEMIR | 78231 |
| IZMIR | YILMAZ | 33515 | KAYA | 22358 | DEMIR | 19304 |
| KONYA | YILMAZ | 14200 | CELIK | 10076 | KAYA | 9673 |
| BURSA | YILMAZ | 27399 | AYDIN | 19775 | OZTURK | 17426 |
| ANKARA | YILMAZ | 47957 | SAHIN | 32057 | OZTURK | 28448 |
| AYDIN | YILMAZ | 14884 | KAYA | 11812 | DEMIR | 11420 |
| KOCAELI | YILMAZ | 16922 | AYDIN | 9795 | KAYA | 9645 |

表 7: Top 3 surnames in most populated cities

From the result table we can see that the most frequently used surnames in top 10 largest city are still Yilmaz, Kaya, Demir, Sahin and so on. It still follows the global pattern and there is no significant relation with the city. This also accord the finding in the name law that surname should only indicate the stemma.

### 4.5 N5. 计算一下该国前 10 大人口城市中，每个城市的人口生日最集中分布的是哪 2 个月

See N3.

### 4.6 N6. 计算前 10 大人口城市人口密度，其中城市的面积可 Google 搜索，面积单位使用平方千米

The city area information are gathered from Wikipedia.

```
city_area = {'ISTANBUL':5343, 'KONYA':38873, 'IZMIR':11891, \
             'ANKARA':24521, 'BURSA':1036,'SIVAS':2768, 'SAMSUN':1055, \
             'AYDIN':1582, 'ADANA':1945, 'SANLIURFA':18584}

for c in city_area:
    print(c, f"\t{pop[c]/city_area[c]:.2f}")
```

From the table we can see that Istanbul is the most densely populated city. No wonder it is the largest city in Turkey.

| City | Density |
|---|---|
| ISTANBUL | 1652.55 |
| KONYA | 34.27 |
| IZMIR | 234.75 |
| ANKARA | 125.67 |
| BURSA | 1722.46 |
| SIVAS | 154.38 |
| SAMSUN | 812.69 |
| AYDIN | 892.31 |
| ADANA | 714.91 |
| SANLIURFA | 41.96 |

表 8: Population density

### 4.7 N7. 根据人口的出身地和居住地，分别统计土耳其跨行政区流动人口和跨城市流动人口占总人口的比例

Use the registration city and district for comparing, because there is only birth city but no birth district.

```
national_pop = dataset.count()
city_mig = dataset.filter(lambda x: x[9]!=x[11]).count()
district_mig = dataset.filter(lambda x: x[10]!=x[12]).count()

print(city_mig / national_pop)
print(district_mig / national_pop)
```

The cross-city migration proportion is 36.14%.
The cross-district migration proportion is 52.39%.

## 5 Hard Problems 将数据按照 70%，10%，20% 的比例分为训练集、验证集和测试集，建模讨论以下问题：

In hard problems, dataframe data structure is used for building models and doing predictions.

```
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.ml.classification import NaiveBayes
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()
dataframe = spark.read.format('csv').option('sep','\t')\
            .option('header','true').load(data_path)
```

### 5.1 H1. 某人所在城市的预测模型：给定一个人的所有信息（除了所在城市），预测这个人所在的城市。分析该模型 Top1 到 Top5 的预测准确度

For this problem, I use the district and neighborhood information for predicting the city. It is reasonable that we can infer the city from district and neighborhood. As mentioned before, the same street name may appear in multiple cities, so I discard it as a feature. For the above two feature, I use a naive Bayes model to solve the problem.

```python
# add label column, the name "label" is by default
label_indexer=StringIndexer(inputCol="address_city",outputCol="label")
# use district and neighborhood as feature
district_indexer = StringIndexer(inputCol="address_district",
                                  outputCol="district_feature")
neighbor_indexer = StringIndexer(inputCol="address_neighborhood",
                                  outputCol="neighbor_feature")
dataframe = label_indexer.fit(dataframe).transform(dataframe)
dataframe = district_indexer.fit(dataframe).transform(dataframe)
dataframe = neighbor_indexer.fit(dataframe).transform(dataframe)
encoder = OneHotEncoder(inputCols=["district_feature","neighbor_feature"],
                        outputCols=["district_vec","neighbor_vec"])
ohe = encoder.fit(dataframe).transform(dataframe)
# the name "features" is by default
assembler = VectorAssembler(inputCols=["district_vec", "neighbor_vec"],
                            outputCol="features")
df = assembler.transform(ohe)

train, valid, test = df.randomSplit([0.7, 0.1, 0.2])
nb = NaiveBayes(smoothing=1.0, modelType="multinomial")
model = nb.fit(train)

y_pred_val = model_h1.transform(valid)
evaluator = MulticlassClassificationEvaluator(labelCol="label",
            predictionCol="prediction", metricName="accuracy")
acc_val = evaluator.evaluate(y_pred_val)
acc_val  # 0.9997794628287017

y_pred_test = model_h1.transform(test)
y_prop_test = y_pred_test.select("probability","label").rdd

def top_K_acc(rows, K):
    in_top_K = lambda row: int(row[1]) in \
               [i[1] for i in
                sorted(list(zip(row[0], range(len(row[0])))), reverse=True)[:K]]
    hit = rows.filter(in_top_K).count()
    tot = rows.count()
    return hit / tot

for K in range(1,6):
    acc = top_K_acc(y_prop_test, K)
    print(f"Top {K} accuracy is {acc}")
```

Top 1 accuracy is 0.9997818256319426
Top 2 accuracy is 0.9999054174830998
Top 3 accuracy is 0.9999992949120146
Top 4 accuracy is 0.9999992949120146
Top 5 accuracy is 0.9999992949120146

From the result we can see that both validation accuracy and test top1 to top5 accuracy are really high. So the naive Bayes model can well perform in this classification task.

## 5.2 H2. 性别预测模型：根据给定一个人的信息（除了性别），预测这个人的性别

As mentioned before, the first name of Turkish is strongly related to gender, while the surname if gender insensitive. So here I use first name as the feature for this classification task. Still use naive Bayes model, though it is degenerated to a maximum likelihood prediction.

```
# add label column
gender_indexer = StringIndexer(inputCol="gender", outputCol="label")
# use first name as feature
name_indexer = StringIndexer(inputCol="first_name", outputCol="name_feature")
dataframe = gender_indexer.fit(dataframe).transform(dataframe)
dataframe = name_indexer.fit(dataframe).transform(dataframe)
encoder = OneHotEncoder(inputCol="name_feature",outputCol="name_vec")
ohe = encoder.fit(dataframe).transform(dataframe)
assembler = VectorAssembler(inputCols=["name_vec"],outputCol="features")
df = assembler.transform(ohe)

train, valid, test = df.randomSplit([0.7, 0.1, 0.2])
nb = NaiveBayes(smoothing=1.0, modelType="multinomial")
model = nb.fit(train)

evaluator = MulticlassClassificationEvaluator(labelCol="label",
            predictionCol="prediction", metricName="accuracy")
y_pred_val = model.transform(valid)
acc_val = evaluator.evaluate(y_pred_val)
acc_val  # 0.9818030460381564
y_pred_test = model.transform(test)
acc_test = evaluator.evaluate(y_pred_test)
acc_test  # 0.9818220031026631
```

Accuracies on both validation set and test set are over 98%, so use first name to predict Turkish gender is very feasible.

## 5.3 H3. 姓名预测模型：假设给定一个人的所有信息（除了姓名），预测这个人最可能的姓氏。分析该模型 Top1 到 Top 5 的预测准确度

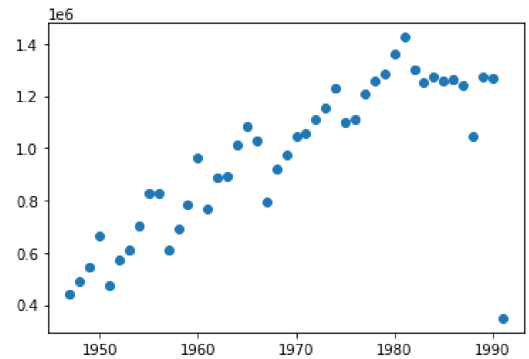
Given the information and analysis above in introduction, E8 and N4, the conclusion can be drawn that there are no feature can be used for surname prediction, in that the surname is only related to stemma but parents' surname are not given in the data.
Therefore I use the most simple and crude method of predicting the most popular K(1-5) surnames of the given person. This model can achieve better performance than other models like naive Bayes or decision tree because all feature in the data are invalid. So these models can never do better than just giving the most popular surnames.

```
train, valid, test = dataset.randomSplit([0.7, 0.1, 0.2])
# find out the most popular surnames in training set
freq_last = train.map(lambda x: (x[3], 1)).reduceByKey(add)\
            .sortBy(lambda x:-x[1]).take(5)
freq_last = [i[0] for i in freq_last]
population = test.count()
for K in range(1, 6):
    hit = test.filter(lambda x:x[3] in freq_last[:K]).count()
    print(f"Top {K} accuracy is {hit/population}")
```

Top 1 accuracy is 0.014247163569189883
Top 2 accuracy is 0.024057654271115295
Top 3 accuracy is 0.03332478634546119
Top 4 accuracy is 0.041534430250531565
Top 5 accuracy is 0.04954783612037707

### 5.4 H4. 人口预测模型：统计每一年出生的人数，预测下一年新增人口数。（使用至少 2 种模型进行预测）

Turkey was established in 1920s.[2] The data before was not trustworthy, so I use the data after 1920 for training and prediction.
The newborns in each year look like this.




First I will use a linear regression model. And for the alternative model, I will use generalized linear model.

```
from pyspark.sql import SQLContext, Row
from pyspark.ml.regression import LinearRegression

year_cnt = dataset.map(lambda x: (int((x[8].split("/"))[2]), 1))\
           .reduceByKey(add).sortBy(lambda x:x[0]).filter(lambda x:x[0]>=1920)\
           .map(lambda row: Row(features=Vectors.dense(row[0]), label=row[1])).toDF()

train, valid, test = year_cnt.randomSplit([0.7, 0.1, 0.2])

# follows the parameter setting in documentation
lr = LinearRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)
model = lr.fit(train)
valid_pred = model.transform(valid)
test_pred = model.transform(test)

valid_pred.show()
test_pred.show()
```

| Year | Truth | Prediction |
|---|---|---|
| 1931 | 199941 | 223569.53 |
| 1932 | 254903 | 242290.63 |
| 1936 | 249897 | 317175.01 |
| 1943 | 346370 | 448222.67 |
| 1961 | 765783 | 785202.38 |
| 1966 | 1027736 | 878807.85 |
| 1975 | 1097350 | 1047297.71 |
| 1989 | 1271867 | 1309393.04 |

表 9: Linear Regression for newborns on validation data

[2]https://en.wikipedia.org/wiki/History_of_Turkey

| Year | Truth | Prediction |
|---|---|---|
| 1920 | 25588 | 17637.49 |
| 1927 | 135264 | 148685.15 |
| 1941 | 306263 | 410780.48 |
| 1958 | 688826 | 729039.09 |
| 1962 | 885179 | 803923.47 |
| 1968 | 917911 | 916250.04 |
| 1970 | 1045109 | 953692.23 |
| 1977 | 1208954 | 1084739.89 |
| 1978 | 1256054 | 1103460.99 |
| 1982 | 1298211 | 1178345.37 |
| 1985 | 1258091 | 1234508.66 |

表 10: Linear Regression for newborns on test data

The following code is the second model.

```
from pyspark.ml.regression import GeneralizedLinearRegression
# follows the parameter setting in documentation
glr = GeneralizedLinearRegression(family="gaussian", link="identity",
                                  maxIter=10, regParam=0.3)

model = glr.fit(train)
valid_pred = model.transform(valid)
test_pred = model.transform(test)

valid_pred.show()
test_pred.show()
```

| Year | Truth | Prediction |
|---|---|---|
| 1931.0 | 199941 | 223569.51 |
| 1932.0 | 254903 | 242290.60 |
| 1936.0 | 249897 | 317174.99 |
| 1943.0 | 346370 | 448222.66 |
| 1961.0 | 765783 | 785202.39 |
| 1966.0 | 1027736 | 878807.86 |
| 1975.0 | 1097350 | 1047297.73 |
| 1989.0 | 1271867 | 1309393.07 |

表 11: Generalized Linear Model for newborns on validation data

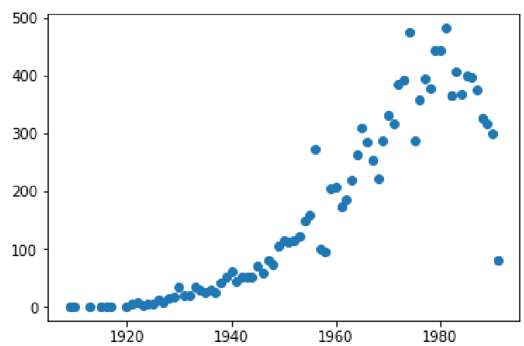
| Year | Truth | Prediction |
|---|---|---|
| 1920 | 25588 | 17637.45 |
| 1927 | 135264 | 148685.12 |
| 1941 | 306263 | 410780.47 |
| 1958 | 688826 | 729039.10 |
| 1962 | 885179 | 803923.48 |
| 1968 | 917911 | 916250.06 |
| 1970 | 1045109 | 953692.25 |
| 1977 | 1208954 | 1084739.92 |
| 1978 | 1256054 | 1103461.02 |
| 1982 | 1298211 | 1178345.40 |
| 1985 | 1258091 | 1234508.69 |

表 12: Generalized Linear Model for newborns on test data

We can see that these two models give out similar predictions.

### 5.5 人又流动预测：统计每年 MALATYA KULUNCAK 这个城市跨城市流动人口，预测下一年跨城市流动人口数

First, this is not a city but a district in a city and there is no second.
The trend looks like this.




Still use the data after 1920 for the same reason as last problem. And I will employ linear regression model for this task.

```
from pyspark.sql import SQLContext, Row

city_diff = dataset.filter(lambda row: (row[9]=="MALATYA" and row[10]=="KULUNCAK")\
            ^(row[11]=="MALATYA" and row[12]=="KULUNCAK"))\
            .map(lambda row: (int(row[8].split("/")[2]), 1)).reduceByKey(add)

mig_year_cnt = city_diff.sortBy(lambda x:x[0]).filter(lambda x:x[0]>=1920)\
               .map(lambda row: Row(features=Vectors.dense(row[0]), label=row[1])).toDF()

train, valid, test = mig_year_cnt.randomSplit([0.7, 0.1, 0.2])
from pyspark.ml.regression import LinearRegression
lr = LinearRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)
model = lr.fit(train)
valid_pred_h5 = model_h5.transform(valid)
test_pred_h5 = model_h5.transform(test)
valid_pred_h5.show()
test_pred_h5.show()
```

| Year | Truth | Prediction |
|---|---|---|
| 1933 | 34 | 27.73 |
| 1935 | 24 | 42.20 |
| 1952 | 116 | 165.19 |
| 1965 | 310 | 259.24 |
| 1967 | 254 | 273.71 |
| 1969 | 288 | 288.18 |

表 13: Linear Regression for migrations on validation data

| Year | Truth | Prediction |
|---|---|---|
| 1924 | 6 | -37.38 |
| 1927 | 8 | -15.67 |
| 1929 | 17 | -1.20 |
| 1946 | 58 | 121.78 |
| 1948 | 74 | 136.25 |
| 1954 | 150 | 179.65 |
| 1959 | 206 | 215.83 |
| 1962 | 186 | 237.53 |
| 1983 | 406 | 389.46 |
| 1985 | 399 | 403.93 |
| 1989 | 318 | 432.86 |
| 1990 | 301 | 440.10 |
| 1991 | 82 | 447.33 |

表 14: Linear Regression for migrations on test data