databricks CreateRawData

(https://databricks.com)

```
%pip install Faker
```

```
%run ./SetupLab
```

Data Generation

```
from pyspark.sql import functions as F
from faker import Faker
from collections import OrderedDict
import uuid
import random
from datetime import datetime, timedelta
import re
import numpy as np
import matplotlib.pyplot as plt
from pyspark.sql.functions import col

def cleanup_folder(path):
  for f in dbutils.fs.ls(path):
    if f.name.startswith('_committed') or f.name.startswith('_started') or f.name.startswith('_SUCCESS') :
      dbutils.fs.rm(f.path)

def fake_date_between(months=0):
  start = datetime.now() - timedelta(days=30*months)
  return F.udf(lambda: fake.date_between_dates(date_start=start, date_end=start + timedelta(days=30)).strftime("%m-%d-%Y %H:%M:%S"))

fake = Faker()
fake_firstname = F.udf(fake.first_name)
fake_lastname = F.udf(fake.last_name)
fake_email = F.udf(fake.ascii_company_email)
fake_date = F.udf(lambda:fake.date_time_this_month().strftime("%m-%d-%Y %H:%M:%S"))
fake_date_old = F.udf(lambda:fake.date_between_dates(date_start=datetime(2012,1,1), date_end=datetime(2015,12,31)).strftime("%m-%d-%Y
%H:%M:%S"))
fake_address = F.udf(fake.address)
channel = OrderedDict([("WEBAPP", 0.5),("MOBILE", 0.1),("PHONE", 0.3),(None, 0.01)])
fake_channel = F.udf(lambda:fake.random_elements(elements=channel, length=1)[0])
fake_id = F.udf(lambda: str(uuid.uuid4()) if random.uniform(0, 1) < 0.98 else None)
countries = ['FR', 'USA', 'SPAIN']
fake_country = F.udf(lambda: countries[random.randint(0,2)])

def get_df(size, month):
  df = spark.range(0, size).repartition(10)
  df = df.withColumn("id", fake_id())
```

```
  df = df.withColumn("id", fake_id())
  df = df.withColumn("firstname", fake_firstname())
  df = df.withColumn("lastname", fake_lastname())
  df = df.withColumn("email", fake_email())
  df = df.withColumn("address", fake_address())
  df = df.withColumn("channel", fake_channel())
  df = df.withColumn("country", fake_country())
  df = df.withColumn("creation_date", fake_date_between(month)())
  df = df.withColumn("last_activity_date", fake_date())
  df = df.withColumn("gender", F.round(F.rand()+0.2))
  return df.withColumn("age_group", F.round(F.rand()*10))


def generateRawData():
    df_customers = get_df(133, 12*30).withColumn("creation_date", fake_date_old())
    for i in range(1, 24):
        df_customers = df_customers.union(get_df(2000+i*200, 24-i))

    df_customers = df_customers.cache()

    ids = df_customers.select("id").collect()
    ids = [r["id"] for r in ids]

    #Number of order per customer to generate a nicely distributed dataset
    np.random.seed(0)
    mu, sigma = 3, 2 # mean and standard deviation
    s = np.random.normal(mu, sigma, int(len(ids)))
    s = [i if i > 0 else 0 for i in s]

    #Most of our customers have ~3 orders
    count, bins, ignored = plt.hist(s, 30, density=False)
    plt.show()
    s = [int(i) for i in s]

    order_user_ids = list()
    action_user_ids = list()
    for i, id in enumerate(ids):
        for j in range(1, s[i]):
            order_user_ids.append(id)
            #Let's make 5 more actions per order (5 click on the website to buy something)
            for i in range(1, 5):
```

```
        for j in range(1, 5):
            action_user_ids.append(id)

print(f"Generated {len(order_user_ids)} orders and  {len(action_user_ids)} actions for {len(ids)} users")

# ORDERS DATA
orders = spark.createDataFrame([(i,) for i in order_user_ids], ['user_id'])
orders = orders.withColumn("id", fake_id())
orders = orders.withColumn("transaction_date", fake_date())
orders = orders.withColumn("item_count", F.round(F.rand()*2)+1)
orders = orders.withColumn("amount", F.col("item_count")*F.round(F.rand()*30+10))
orders = orders.cache()
orders.repartition(10).write.format("json").mode("overwrite").save(rawDataDirectory+"/orders")
cleanup_folder(rawDataDirectory+"/orders")

# WEBSITE ACTIONS DATA
platform = OrderedDict([("ios", 0.5),("android", 0.1),("other", 0.3),(None, 0.01)])
fake_platform = F.udf(lambda:fake.random_elements(elements=platform, length=1)[0])

action_type = OrderedDict([("view", 0.5),("log", 0.1),("click", 0.3),(None, 0.01)])
fake_action = F.udf(lambda:fake.random_elements(elements=action_type, length=1)[0])
fake_uri = F.udf(lambda:re.sub(r'https?:\/\/.*?\/', "https://databricks.com/", fake.uri()))

actions = spark.createDataFrame([(i,) for i in order_user_ids], ['user_id']).repartition(20)
actions = actions.withColumn("event_id", fake_id())
actions = actions.withColumn("platform", fake_platform())
actions = actions.withColumn("date", fake_date())
actions = actions.withColumn("action", fake_action())
actions = actions.withColumn("session_id", fake_id())
actions = actions.withColumn("url", fake_uri())
actions = actions.cache()
actions.write.format("csv").option("header", True).mode("overwrite").save(rawDataDirectory+"/events")
cleanup_folder(rawDataDirectory+"/events")

# CHURN COMPUTATION AND USER GENERATION

#Let's generate the Churn information. We'll fake it based on the existing data & let our ML model learn it
churn_proba_action = actions.groupBy('user_id').agg({'platform': 'first', '*': 'count'}).withColumnRenamed("count(1)", "action_count")
#Let's count how many order we have per customer.
churn_proba = orders.groupBy('user_id').agg({'item_count': 'sum', '*': 'count'})
churn_proba = churn_proba.join(churn_proba_action, ['user_id'])
```

```
churn_proba = churn_proba.join(churn_proba_action, ['user_id'])
churn_proba = churn_proba.join(df_customers, churn_proba.user_id == df_customers.id)

#Customer having > 5 orders are likely to churn

churn_proba = (churn_proba.withColumn("churn_proba", 5 +  F.when(((col("count(1)") >=5) & (col("first(platform)") == "ios")) |
                                                                 ((col("count(1)") ==3) & (col("gender") == 0)) |
                                                                 ((col("count(1)") ==2) & (col("gender") == 1) & (col("age_group") <= 3)) |
                                                                 ((col("sum(item_count)") <=1) & (col("first(platform)") == "android")) |
                                                                 ((col("sum(item_count)") >=10) & (col("first(platform)") == "ios")) |
                                                                 (col("action_count") >=4) |
                                                                 (col("country") == "USA") |
                                                                 ((F.datediff(F.current_timestamp(), col("creation_date")) >= 90)) |
                                                                 ((col("age_group") >= 7) & (col("gender") == 0)) |
                                                                 ((col("age_group") <= 2) & (col("gender") == 1)), 80).otherwise(20)))

churn_proba = churn_proba.withColumn("churn", F.rand()*100 < col("churn_proba"))
churn_proba = churn_proba.drop("user_id", "churn_proba", "sum(item_count)", "count(1)", "first(platform)", "action_count")
churn_proba.repartition(100).write.format("json").mode("overwrite").save(rawDataDirectory+"/users")
cleanup_folder(rawDataDirectory+"/users")
```