eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

# Informatika Fakultatea

## Adimen Artifizialeko Gradua

DPMA - Praktika 2

---

Ekhiñe Irigoyen eta Garazi Garrido

2025/05/15

# Aurkibidea

# 1. Sarrera

Gaur egun, sare sozialek gure bizitzetan duten eragina ukaezina da. Milioika pertsona konektatzen dira egunero plataformetara, informazioa partekatzeko, besteekin harremanak izateko edota iritzia emateko. Ingurune digital honetan, erabiltzaileek erakusten duten jokabideak eta sarean dituzten harremanek gizarte-dinamiken inguruko informazio baliotsua eskaintzen dute. Hori kontuan hartuta, lan honen bidez Twitter sare sozialeko erabiltzaileen informazioa aztertu ahal izateko azpiegitura diseinatzen lan egin dugu, haien pertsonalitateak oinarri hartuta.

Lana entregatzeko arazoak izan ditugunez, pisu handia duelako, github-era igo dugu. Link-a: https://github.com/ekhinee/lana3_garazi_ekhine.git . Github-en ere memoria atxikitu dugu.

# 2. Soluzioaren azalpena

## 2.1 Lerro orokorrak

Lan hau aurrera eramateko, hainbat tresna eta teknologia erabili dira, datuen tratamendu osoa automatizatutako pipeline batean integratuz. Hasieran, lau fitxategi jaso dira, 3 JSON eta 1 CSV formatuan: erabiltzaileen informazioa, sareko harremanak eta txioak jasotzen dituztenak. CSV-kok datua MySQL-n bolkatu da, eta Python script baten bidez MQTT erabiliz txioen publikazioa automatizatu da. Ondoren, zenbait hobekuntza tarteko, PostgreSQL-era migratu dira.

Kafka erabili da denbora errealeko txioak sisteman txertatzeko, eta Flink SQL bidez prozesatu dira, pertsonalitate moten arabera txioen banaketa aztertuz. Hasierako saiakeretan Elasticsearch erabili da emaitzak gordetzeko eta Kibanan bistaratzea lortzeko, baina ondoren PostgreSQL-era migratu da irteera, eta Apache Superset aukeratu da azken bistaratze-tresna gisa, informazioa modu grafiko, garbi eta interaktiboan eskaintzeko.

Horrela, pipeline oso bat osatu da: datuen hasierako karga eta transformazioa (MySQL eta Python bidez), streaming datuen kudeaketa (Kafka), denbora errealeko analisia (Flink SQL), eta azkenean emaitzen gordetzea eta bistaratzea (Elasticsearch/PostgreSQL + Kibana/Superset). Prozesu guztiak automatikoki lotuta eta sinkronizatuta daude, sistemaren egonkortasuna eta eskalagarritasuna bermatuz.

## 2.2 Fitxategiak

Behin azaldu dugula lanaren nondik norakoak, azter dezagun hau guztia martxan jarri ahal izateko behar izan ditugun fitxategi multzoa. Honako hauek dira soluzioa osatzen duten fitxategi eta karpetak:

- `data/`

```
        ○ users1.json
        ○ tweets1.json
        ○ edges1.json
        ○ mbti_labels.csv
    ● jars/ #kafka-rako
    ● jars_flink/ #flink-erako
    ● docker-compose.yml
    ● connect-mqtt-source.json
    ● Dockerfile.python
    ● Dockerfile.superset
    ● flink-conf.yaml
    ● init.sql
    ● load_json_to_mysql.py
    ● requirements_python.txt
    ● superset_config.py
    ● tweets_create.py
```

➢ **data/**

Lan honetan erabilitako datuak lau fitxategitan banatuta daude: hiru json formatuan eta bat csv formatuan. Fitxategi hauek Twitter sare sozialeko erabiltzaileen informazioa beraien arteko harremanak, txioak eta pertsonalitate psikologikoari buruzko etiketak jasotzen dituzte. Jarraian, fitxategi bakoitza xehetasunez deskribatzen da:

- **users1.json**: Fitxategi honek Twitterreko erabiltzaileei buruzko informazio orokorra gordetzen du: id, screen_name, description, location, followers_count, friends_count… eta abar. Informazio horren bidez, erabiltzaileen profila osatu eta beraien osagarrien inguruko analisiak egin daitezke.
- **edges1.json**: Fitxategi honek sare sozialaren egitura modelatzeko beharrezkoa den informazioa dauka. Zehazki, erabiltzaileen arteko jarraipen-harremanak adierazten dira, source_id eta targer_id eremuak erabiliz.
- **tweets1.json**: Fitxategi honetan jasotzen dira erabiltzaileek idatzitako txioak. Txio bakoitzeko user_id, created_at, lang, retweet_count bezalako informazioa eskeintzen da.
- **mbti_labels.csv**: Fitxategi honek erabiltzaileei atxikitako MBTI (Myers–Briggs Type Indicator) pertsonalitate etiketak biltzen ditu. user_id eta mbti_type eremuak ditu, pers_id-rekin batera. Hala, erabiltzaile bakoitzaren MBTI pertsonalitatea jakin dezakegu.

Datu horiek guztiek osatzen dute lan honen oinarria. Sare sozial batean oinarritutako testuinguruan, informazio hau aberatsa eta askotarikoa da, eta aukera ematen du ikuspegi estatistiko, linguistiko eta sozialetik azterketa sakonak egiteko.

- ➢ **`jars/ eta jars_flink/`**

Hemen proiektua garatzeko beharrezkoak izan zaizkigun jar fitxategiak aurkituko ditugu. *jars* karpetan kafkari dagozkienak izango ditugu eta *jars_flink* karpetan berriz, flink-i lotutako jar fitxategiak.

- ➢ **`docker-compose.yml:`**

Fitxategi honetan, zerbitzu ugariz osatutako ingurune bat definitzen da, datuen ingesta, transformazioa eta bistaratzea integratzen dituena. Zerbitzu bakoitzak funtzio espezifiko bat betetzen du, eta guztiak lana3_net sarearen bidez konektatzen dira. Honako hauek dira zerbitzu nagusiak:

- MySQL: MySQL zerbitzari bat exekutatzen duen kontainerra da. Bertan, init.sql script-aren bidez hasierako taulak eta datuak sortzen dira, zehazki mbti_labels.csv fitxategia “bolkatzen” da mysql taula batean.
- Mosquitto: MQTT protokolo arina erabiliz mezuak kudeatzen dituen zerbitzaria. Datuen fluxuak kontrolatzeko erabiltzen da, batez ere python kontainerretik bidalitako mezuak kafkara igarotzeko.
- python: Dockerfile berezi batekin sortutako kontainerra da. Python script-ak exekutatzeko erabiltzen da, adibidez MQTT protokoloaren bidez mosquitton publikatutako txioak kafkan publikatzeko.
- Zookeeper eta Kafka: Zookeeper-ek Kafka-ren koordinazioa kudeatzen du eta Kafka-k mezuen streaming plataforma gisa funtzionatzen du. connect-custom izeneko topic bat sortzen da automatikoki. Kafka eta Flink-en arteko lotura funtsezkoa da datuen eraldaketa denbora errealean egiteko.
- Kafka-Connect: Kafka eta beste sistemaren arteko konektoreak kudeatzen ditu. Pluginen direktorio bat muntatzen da, bertan konektore pertsonalizatuak jartzeko.
- Flink (JobManager eta TaskManager): Flink-ek datuen eraldaketa eta analisi denbora errealean egiten du. SQL kontsulten bidez, Kafka, MySQL, PostgreSQL eta Elasticsearch zerbitzuen arteko fluxuak kudeatzen dira. JAR fitxategi ugari muntatzen dira konektoreak gaitzeko: Kafka, MySQL, PostgreSQL eta Elasticsearch.
- Elasticsearch eta Kibana: Elasticsearch-ek indexatutako dokumentuak gordetzen ditu eta Kibana-k horiek bisualizatzeko interfazea eskaintzen du. Flink-ek sortutako aggregazioak Elasticsearch-era bidaltzen ziren hasieran, baina proiektuaren azken fasean PostgreSQL eta Superset erabiltzen dira.
- PostgreSQL: Azken biltegiratze-plataforma gisa erabiltzen da. Flink-en bidez sortutako datuak PostgreSQLra bidaltzen dira, eta hortik Superset-ek hartzen ditu.
- Superset: Elasticsearch zein PostgreSQLko datuak grafiko interaktiboen bidez bistaratzeko erabiltzen da. Dockerfile propio bat erabiliz eraikitzen da eta konfigurazio fitxategi pertsonalizatu bat montatzen da.

Azken finean, docker-compose.yml fitxategi honek ingesta → biltegiratze → prozesatze → bistaratze pipeline oso bat eskaintzen du, teknologia ugari integratuz eta datu-analisi proiektu konplexuak egiteko prest.

- ➢ **`connect-mqtt-source.json:`**

connect-mqtt-source.json fitxategiak Kafka Connect-en bidez MQTT iturri batetik datuak inportatzeko konfigurazioa biltzen du. Kasu honetan, mosquitto izeneko MQTT zerbitzuarekin konektatzen da, eta bertan tweets izeneko gaia jarraitzen du. Gai horretako mezuak connect-custom izeneko Kafka topikoan sartzen dira. Konektorea MqttSourceConnector klasean oinarritzen da, eta mezuak byte array modura jasotzen dira, ByteArrayConverter erabilita. Kafka Connect-ek kafka:9094 helbidea erabiltzen du Kafka zerbitzuarekin konektatzeko, eta ez da erreplikazio handirik behar, beraz replication.factor 1ean ezartzen da. Konfigurazio honekin, sistema gai da IoT edo txio fluxu arinetatik datozen datuak modu errazean Kafka-ra igarotzeko.

- **Dockerfile.python:**

Dockerfile.python fitxategiak Python ingurune arin eta pertsonalizatu bat sortzen du. Oinarri moduan python:3.9-slim irudia erabiltzen da. /data direktorioa ezartzen da lanerako direktorio gisa, eta bertara kopiatzen dira bai Python script-ak (tweets_create.py, load_json_to_mysql.py), bai JSON datu-fitxategiak (tweets1.json, edges1.json, users1.json) eta dependentzietarako fitxategia (requirements_python.txt). Azkenik, tweets_create.py script-a exekutatzen da kontainerrak abiatzean, horrela automatizatuz datuen tratamendua edo karga.

- **Dockerfile.superset:**

Dockerfile.superset fitxategiak Superset ingurunea zabaltzen du datu-iturburu ezberdinekin lan egiteko gaitasun handiagoa emanez. Oinarrian apache/superset:latest irudia erabiltzen da. Beraz, fitxategi honen bitartez gure superset kontainerra abiarazten da.

- **init.sql:**

init.sql fitxategiak MySQL datu-basea hasieratzeko script modura funtzionatzen du. Lehenik eta behin, mbtidb izeneko datu-basea sortzen du existitzen ez bada, eta user erabiltzaileari zein root-i baimen osoak ematen dizkio, urruneko sarbidea ahalbidetuz. Ondoren, mbti_labels izeneko taula sortzen da, non MBTI nortasun-etiketak eta identifikadore bakarrak gordetzen diren. Azkenik, LOAD DATA INFILE aginduaren bidez, /data/mbti_labels.csv fitxategitik datuak inportatzen dira automatikoki, analisiari ekiteko beharrezko informazioa datu-basean eskuragarri egon dadin.

- **load_json_to_mysql.py:**

Pythoneko script honek users1.json eta edges1.json fitxategietako datuak MySQL datu-base batean kargatzen ditu. Lehenik, users taula eta edges taula sortzen dira erabiltzaileei eta haien jarraipenei buruzko informazioa gordetzeko. Ondoren, JSON fitxategietako edukia irakurri eta dagokien tauletan txertatzen da, lehendik existitzen diren erregistroak bikoiztu gabe. Azkenik, followers_count izeneko taula bat sortzen da, eta bertan erabiltzaile bakoitzak zenbat jarraitzaile dituen kalkulatzen da, is_followed_by eremua aztertuz. Script honek datuen hasierako prestaketa automatizatzen du, ondorengo analisiak eta bistaratzeak errazteko.

- **requirements_python.txt:**

Fitxategi honetan gure python kontainerrerako beharrezkoak izango diren instalazioak jartzen dira.

- ➢ **`superset.config.py:`**

Fitxategi honek Superset-en konfigurazio pertsonalizatu bat definitzen du. SECRET_KEY aldagaia ezartzen da, aplikazioaren segurtasunarekin lotutako funtzionalitateetarako (adibidez, saioen kudeaketa) erabiliko den gakoa izanik. Gainera, pymysql moduluaren bidez MySQL zerbitzarietarako euskarria aktibatzen da, Supersetek MySQL datu-baseekin modu bateragarrian lan egin dezan.

- ➢ **`tweets_create.py:`**

Script honek tweets1.json fitxategian dauden txioak hartzen ditu eta bi segunduro behin, ausazko erabiltzaileren batek idatzitako txio bat bidaltzen du MQTT bidez tweets izeneko gai batera. paho.mqtt.client liburutegiaren bidez konektatzen da mosquitto izeneko MQTT zerbitzarira, eta txio bakoitza JSON formatuan prestatzen da, erabiltzailearen IDa, txioaren testua eta bidalketaren ordua barne. Horrela, datu-fluxu erreala simulatzen da ondoren Kafka ingurune batean erabiltzeko.

## 3. Hasieratzeko pausuak

Soluzioa martxan jartzeko honako pausu hauek jarraitu behar dira:

### 3.1. Kontainerrak martxan jarri

Lehenik eta behin, **`docker-compose up -d –build`** komandoa exekutatu behar da konfiguratutako kontainer guztiak martxan jartzeko. Komando honek *mysql*, *mosquitto*, *python*, *zookeeper*, *kafka, kafka-connect*, *jobmanager, taskmanager, elasticsearch, kibana postgres* eta *superset* kontainerrak abiaraziko ditu, bakoitza bere ingurunearekin.

```
docker-compose up -d --build
```

### 3.2. Mosquitto eta kafka konektatu

Aurretik esplikatu dugun moduan, tweets_create.py script-aren bidez, ausaz ./data/tweets1.json-etik mezuak publikatzen joango dira mosquitton. Oraingo pausuan mosquitto eta kafka konektatuko ditugu, denbora errealean lana egiteko balio digun tresna, eta kafkan publikatuko dira mezu horiek. Horretarako kafka-connect erabiltzen dugu, zeinak bi herramientak (mosquitto eta kafka) konektatzen lagunduko digun. Konexioa gauzatzeko fitxategi extra bat beharko dugu, mqtt-source.json, zeinak konexioa gauzatzeko beharrezko konfigurazioak dituen. Terminalean curl baten bidez eramango dugu aurrera hau guztia, baina komandoa jaurti aurretik itxaron egin behar dugu unetxo bat.

```
curl -d @connect-mqtt-source.json -H "Content-Type: application/json" -X POST http://localhost:8083/connectors
```

Kafkan txioak egoki publikatzen direla ikusi nahi izanez gero, ondoko komando hau jaurti dezakegu terminalean bertan:

```
docker exec kafka kafka-console-consumer \
  --bootstrap-server kafka:9094 \
  --topic connect-custom \
  --from-beginning
```

Atera beharko litzatekeen emaitza honako itxurakoa da:

{"user_id": 21055439, "tweet": "RT @TheMaineLBot: Just a reminder that if you play Identify by @themaine at exactly 11:59:23 pm, you'll start the New Year with 3, 2, 1,\u2026.", "timestamp": "2025-05-12 08:18:17"}
…

### 3.3. Taulak sortu flink-en

Behin txioak egoki publikatzen ari direla egiaztatu ondoren, flink sar dezakegu jokoan. Gure helburua iristen diren txioen erabiltzaileen mbti pertsonalitateak aztertu eta kontatzea da. Horretarako hainbat taula sortu beharko ditugu. Hasteko flink-era konektatu behar gara. Flink *jobmanager* eta *taskmanager*-rrek osatzen dute. Gu *jobmanager* kontainerrera sartuko gara lehendabizi `docker exec -it jobmanager bash` komandoaren bitartez. Behin barruan gaudela flink-era konektatu gaitezke ondoko komando honekin: `/opt/flink/bin/sql-client.sh -l /opt/flink/lib`.

```
docker exec -it jobmanager bash

/opt/flink/bin/sql-client.sh -l /opt/flink/lib
```

Flink-era konektatuta gaudela batetik mysql-n dugun taula flink-en sortu beharko dugu, eta bestetik txioei dagozkiena. Ondoren bi hauek join batekin elkartuko ditugu hirugarren taula

bat sortuz, zeina elasticsearch-en biltegiratuko dugun insert bat eginez. Ondoko lau komandoak jaurti behar dira guzti hau egiteko.

```
# mbti_labels taula mysql-tik hartua
CREATE TABLE personalities (
      id BIGINT,
      mbti_personality STRING,
      pers_id INT
) WITH (
      'connector' = 'jdbc',
      'url' = 'jdbc:mysql://mysql:3306/mbtidb?useSSL=false',
      'table-name' = 'mbti_labels',
      'driver' = 'com.mysql.jdbc.Driver',
      'username' = 'root',
      'password' = 'root'
);


# kafkan publikatzen diren txioak
CREATE TABLE txioak (
  user_id BIGINT,
  tweet STRING,
  `timestamp` STRING,
  ts AS TO_TIMESTAMP(`timestamp`),  -- Campo computado
  WATERMARK FOR ts AS ts - INTERVAL '5' SECOND
) WITH (
  'connector' = 'kafka',
  'topic' = 'connect-custom',
  'properties.bootstrap.servers' = 'kafka:9094',
  'properties.group.id' = 'flink-group',
  'scan.startup.mode' = 'earliest-offset',
  'format' = 'json',
  'json.fail-on-missing-field' = 'false',
  'json.ignore-parse-errors' = 'true'
);


#bi taulak elkartu eta elasticsearch-en gorde
CREATE TABLE count_per_personality (
  mbti_label STRING,
  cnt BIGINT,
  mbti_index INT,
  PRIMARY KEY (mbti_label, mbti_index) NOT ENFORCED
) WITH (
  'connector' = 'elasticsearch-7',
  'hosts' = 'http://elasticsearch:9200',
  'index' = 'count_per_personality',
  'document-id.key-delimiter' = '_',
  'format' = 'json',
  'sink.bulk-flush.max-actions' = '1'
);

# datuak gehitu taulan
INSERT INTO count_per_personality
SELECT
  p.mbti_personality AS mbti_label,
  COUNT(*) AS cnt,
  p.pers_id AS mbti_index
FROM txioak t
```

```
LEFT JOIN personalities p ON t.user_id = p.id
GROUP BY
  TUMBLE(t.ts, INTERVAL '1' MINUTE),
  p.mbti_personality,
  p.pers_id;
```

Taulak sortu eta bete ondoren, elasticsearch-en gordetako taulari dagokion indizea sortuko da, behin datuak sartu direnean. Sortu dugun taula denbora errealean, minutu bateko *window*-arekin joango da eguneratzen. Hau guztia egin ondoren *kibana*-n bistaratu dezakegu iristen zaigun informazio hori.

### 3.4. Kibana konfiguratu

Kibana martxan jarri eta konfiguratu behar da, ***http://localhost:5601*** helbidean sartuz. Kibana konfiguratzeko:

**Index Patterns** sakatu -> **Create Index Pattern** sakatu -> **count_per_personality** idatzi -> **Next Step** sakatu -> **create index pattern** sakatu

Orain hasierara bueltatuko gara (Home)

**Visualize** sakatu -> **Create new visualization** sakatu -> **Pie** aukeratu -> **Count_per_personality** aukeratu

Jarraian grafikoaren konfigurazioa ezarriko dugu:

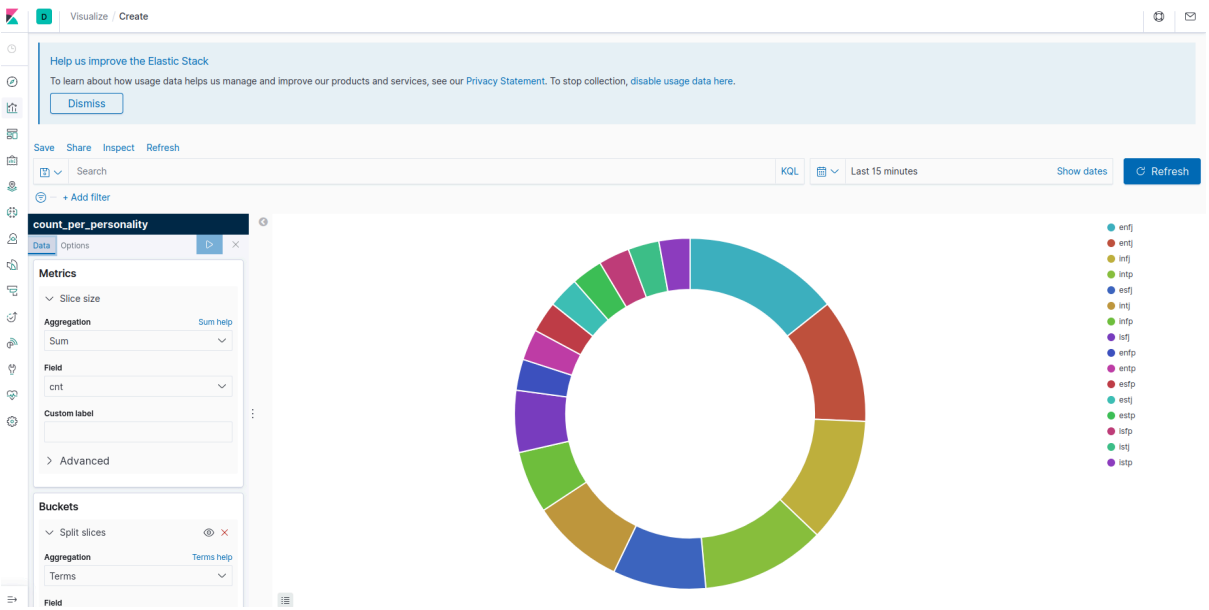
**Metrics** atalean:
- Aggregation: Sum
- Field: cnt

**Buckets** atalean (**Add** sakatu -> **Split slices** aukeratu):
- Aggregation: Terms
- Field: mbti_label.keyword
- Size: 16

Hori guztia egin ondoren **Apply changes** saka dezakegu (konfigurazioaren goiko eskuineko aldean dagoen gezia).

## 4.Soluzioaren funtzionamendua

Hori guztia ongi konfiguratu ondoren, horrelako emaitza bat lortzen da:




Kontuan izan, minuturo eguneratzen doala txioak heltzen diren einean, beraz, **Refresh** egin dezakegu minuturo eta ikusiko dugu nola aldatzen diren.

## 5. Hobekuntzak

### 5.1 Superset: Kibana-ren alternatiba

Superset Apache proiektuko tresna bat da, datu-bisualizazio interaktiborako eta dashboard dinamikoak sortzeko aukera ematen duena. Web interfaz sinple eta intuitibo baten bidez, hainbat datu-iturrirekin konektatu daiteke (PostgreSQL, MySQL, SQLite, eta abar), SQL kontsultak exekutatu eta emaitzak grafiko moduan aurkezteko. Kasu honetan, Superset Kibana-ren alternatiba gisa aurkeztu dugu, azken hau batez ere Elasticsearch-en gainean erabili da, eta Superset PostgreSQL-en gainean erabiliko dugu.

Beraz, lehen pausoa taula PostgreSQL-n jartzea izango da. Horretarako lehendabizi PostgreSQL-ren kontainerrean sartuko gara, eta taula sortuko dugu. Eta ondoren, Flink-en datuak gehituko ditugu. Hala ere, berriz sortu beharko ditugu personalities eta txioak taulak Flink-etik atera bagara lehen, galdu egiten baitira.

```
docker exec -it postgres psql -U postgres -d mbti_analytics

# Sortu taula
CREATE TABLE count_per_personality (
  mbti_label TEXT,
  cnt BIGINT,
  mbti_index INT,
  PRIMARY KEY (mbti_label, mbti_index)
);
```

```
exit

docker exec -it jobmanager bash
/opt/flink/bin/sql-client.sh -l /opt/flink/lib

CREATE TABLE personalities (
      id BIGINT,
      mbti_personality STRING,
      pers_id INT
) WITH (
      'connector' = 'jdbc',
      'url' = 'jdbc:mysql://mysql:3306/mbtidb?useSSL=false',
      'table-name' = 'mbti_labels',
      'driver' = 'com.mysql.jdbc.Driver',
      'username' = 'root',
      'password' = 'root'
);

CREATE TABLE txioak (
  user_id BIGINT,
  tweet STRING,
  `timestamp` STRING,
  ts AS TO_TIMESTAMP(`timestamp`),
  WATERMARK FOR ts AS ts - INTERVAL '5' SECOND
) WITH (
  'connector' = 'kafka',
  'topic' = 'connect-custom',
  'properties.bootstrap.servers' = 'kafka:9094',
  'properties.group.id' = 'flink-group',
  'scan.startup.mode' = 'earliest-offset',
  'format' = 'json',
  'json.fail-on-missing-field' = 'false',
  'json.ignore-parse-errors' = 'true'
);

CREATE TABLE count_per_personality (
  mbti_label STRING,
  cnt BIGINT,
  mbti_index INT,
  PRIMARY KEY (mbti_label, mbti_index) NOT ENFORCED
) WITH (
  'connector' = 'jdbc',
  'url' = 'jdbc:postgresql://postgres:5432/mbti_analytics',
  'table-name' = 'count_per_personality',
  'driver' = 'org.postgresql.Driver',
  'username' = 'postgres',
```

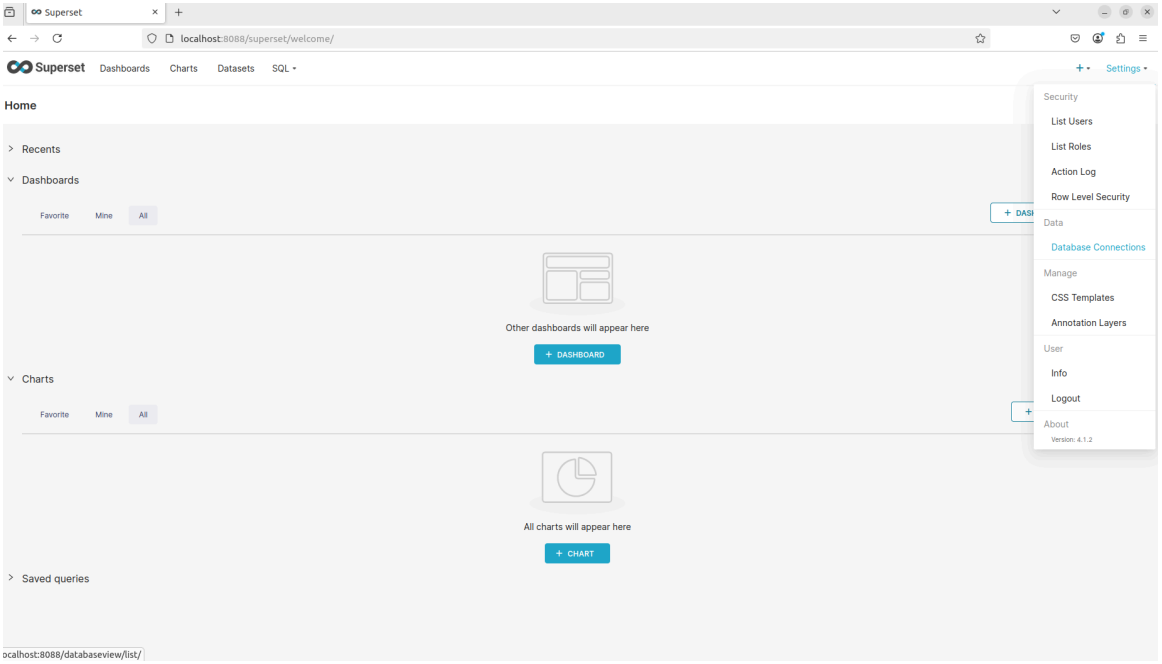
```
  'password' = 'postgres'
);

INSERT INTO count_per_personality
SELECT
  p.mbti_personality AS mbti_label,
  COUNT(*) AS cnt,
  p.pers_id AS mbti_index
FROM txioak t
LEFT JOIN personalities p ON t.user_id = p.id
GROUP BY
  TUMBLE(t.ts, INTERVAL '1' MINUTE),
  p.mbti_personality,
  p.pers_id;
```

Iada prest dugula taula, Superseten bistaratzea izango da hurrengo pausua. Superset abiarazteko zenbait pauso jarraitu behar dira. Hurrengo komandoak terminaleak jarri ondoren prest izango dugu erabiltzeko:

```
docker exec -it superset bash
superset db upgrade

# Sortu admin erabiltzailea
export FLASK_APP=superset
superset fab create-admin \
   --username admin \
   --firstname Admin \
   --lastname User \
   --email admin@example.com \
   --password admin

superset init
```

Behin hori eginik superset-en interfazera joan gaitezke honako hau jarriz nabigatzaile batean: ***http://localhost:8088***. Superseten sartzeko USERNAME: admin eta PASSWORD: admin jarri beharko ditugu.

Bistaratzea egiteko eman beharreko pausuak: (ikusi irudiak laguntzeko).

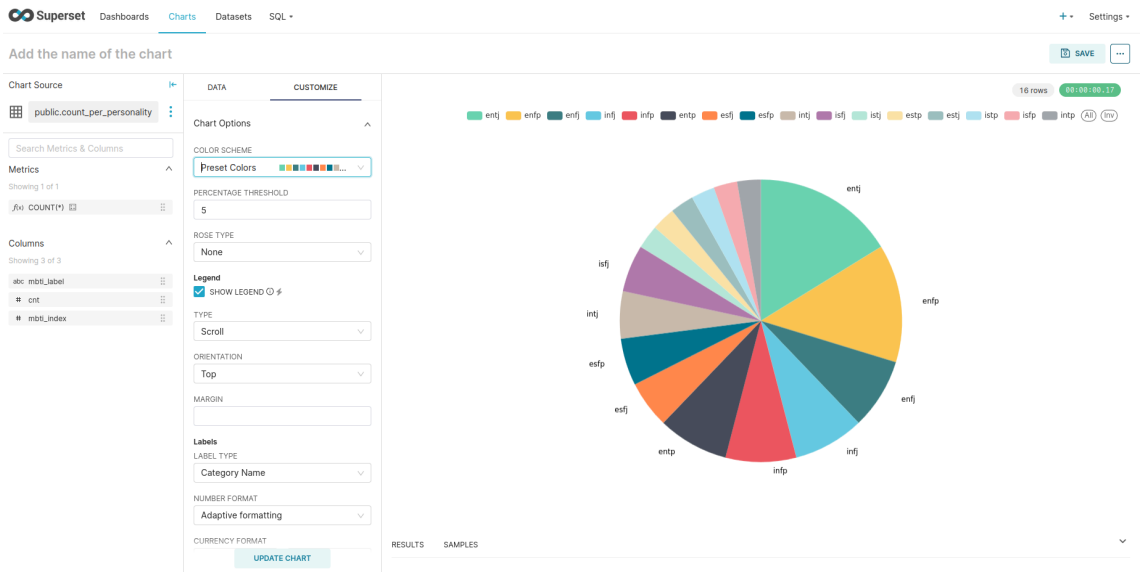



**Settings** -> **Database Connections** -> **+Database** -> **PostgreSQL** -> **connect this database with a SQLAlchemy URI string instead** (behe behean)
- SQLAlchemy URI*: postgresql://postgres:postgres@postgres:5432/mbti_analytics
**Connect** sakatu amaitzeko (Test connection ere saka daiteke ondo doala ikusteko)

**Datasets** -> **+Dataset**
{Database: PostgreSQL, schema: public, table:count_per_personality}
**Create dataset and create chart** -> **Pie Chart** -> **Create new chart**

Orain ikus dezagun nola konfiguratu grafikoa behar bezala ateratzeko:

DATA: {Dimensions: mbti_label, metric: SUM(cnt)}
Lortutako emaitza:



## 5.2 Users1.json eta Edges1.json-en erabilera postgres-en

Jarraian Users1.json eta Edges1.json fitxategietako datuak erabiliko ditugu. Hori egiteko aurrenik mysql-n jarriko dugu haien informazioa taula antolatuetan. Gainera taula extra bat ere erabiliko dugu gerorako beharrezko kontaketa bat egiteko, zehazki erabiltzaile bakoitzaren jarraitzaile kopurua batuko duena. Hori egiteko .py script bat exekutatuko dugu python kontainerrean:

```
docker exec -it python bash
python3 load_json_to_mysql.py
```

Hori egin ondoren, iada mysql-n beharrezko taulak izango ditugu. Berez, hurrengo taulak mysql-n bertan sortu genitzake superset-en bistaratzea erraza delako mysql erabiliz, baina, PostgreSQL-n egitea erabili dugu aurreko ildoa jarraituz. Beraz, lehenik eta behin, PostgreSQL-n sortuko dugu beharko ditugun taulak.

Bi bistaraketa egingo ditugunez, banan banan esplikatuko ditugu, nahiz eta errepikakorragoa izan, argiagoa ere baita.

Has gaitezen radar chart-arekin. Helburua izango da pertsonalitate bakoitzeko jarraitzaile kopurua, retweet kopurua… eta beste datu batzuren arteko erlazioa aztertzea. Hasteko esan bezala PostgreSQL-n beharrezko taula sortuko dugu:

```
docker exec -it postgres psql -U postgres -d mbti_analytics

CREATE TABLE IF NOT EXISTS mbti_stats_for_radar (
      mbti_label TEXT PRIMARY KEY,
      avg_retweets DOUBLE PRECISION,
      avg_favorites DOUBLE PRECISION,
      avg_hashtags DOUBLE PRECISION,
      avg_mentions DOUBLE PRECISION,
      avg_media DOUBLE PRECISION,
      avg_followers DOUBLE PRECISION
);
```

Ondoren flink-en egingo dugu informazio mugimendu guztia, beraz bertara sartu beharko gara, eta beharrezko taulak sortu:

```
docker exec -it jobmanager bash
/opt/flink/bin/sql-client.sh -l /opt/flink/lib
```

Orain batetik mbti_labels taula (lehen personalities deitua), users, edges eta followers_count flinken sortuko ditugu:

```
CREATE TABLE users (
      id BIGINT,
      screen_name STRING,
      location STRING,
      verified BOOLEAN,
      statuses_count INT,
      total_retweet_count INT,
      total_favorite_count INT,
      total_hashtag_count INT,
      total_mentions_count INT,
      total_media_count INT
) WITH (
      'connector' = 'jdbc',
      'url' = 'jdbc:mysql://mysql:3306/mbtidb',
      'table-name' = 'users',
      'username' = 'root',
      'password' = 'root'
);

CREATE TABLE edges (
      id BIGINT,
      follows STRING,
      is_followed_by STRING
) WITH (
      'connector' = 'jdbc',
      'url' = 'jdbc:mysql://mysql:3306/mbtidb',
      'table-name' = 'edges',
      'username' = 'root',
      'password' = 'root'
);

CREATE TABLE mbti_labels (
      id BIGINT,
      mbti_personality STRING,
      pers_id INT
) WITH (
      'connector' = 'jdbc',
      'url' = 'jdbc:mysql://mysql:3306/mbtidb',
      'table-name' = 'mbti_labels',
      'username' = 'root',
      'password' = 'root'
);


CREATE TABLE followers_count (
      user_id BIGINT,
      num_followers INT
```

```
) WITH (
      'connector' = 'jdbc',
      'url' = 'jdbc:mysql://mysql:3306/mbtidb',
      'table-name' = 'followers_count',
      'username' = 'root',
      'password' = 'root'
);
```

Ondoren, desio digun bistaraketa egiteko datuak bilduko dituen taula sortuko dugu, informazio guztia elkartuko duena:

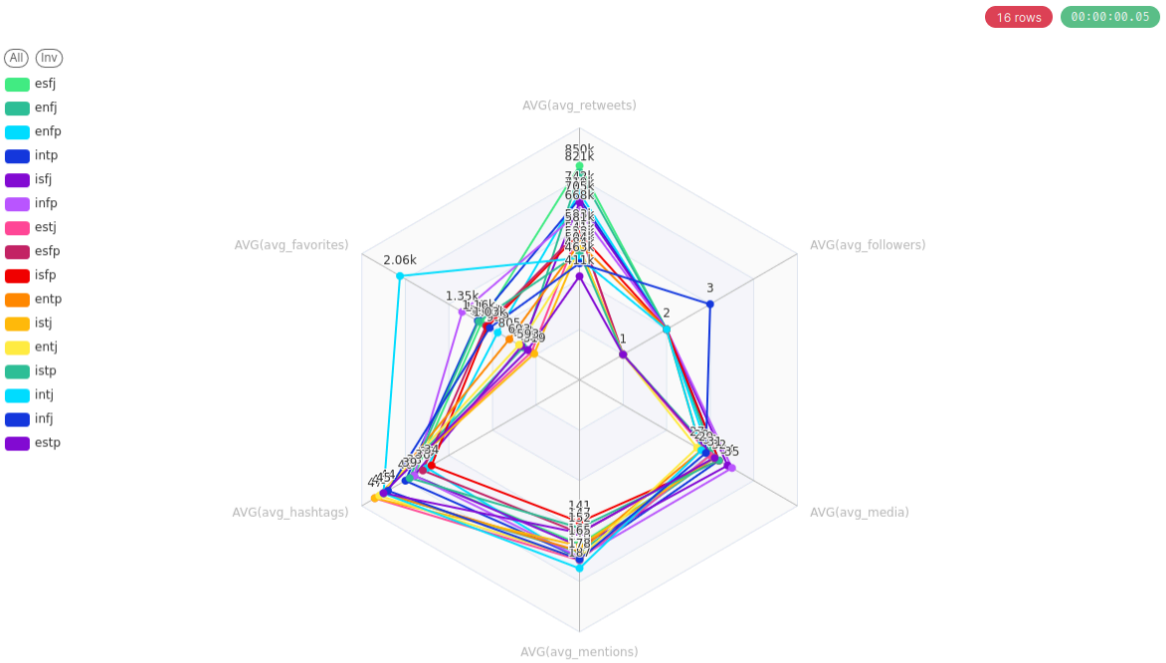
```
CREATE TABLE mbti_stats_for_radar (
      mbti_label STRING,
      avg_retweets DOUBLE,
      avg_favorites DOUBLE,
      avg_hashtags DOUBLE,
      avg_mentions DOUBLE,
      avg_media DOUBLE,
      avg_followers DOUBLE,
      PRIMARY KEY (mbti_label) NOT ENFORCED
) WITH (
      'connector' = 'jdbc',
      'url' = 'jdbc:postgresql://postgres:5432/mbti_analytics',
      'table-name' = 'mbti_stats_for_radar',
      'username' = 'postgres',
      'password' = 'postgres'
);

INSERT INTO mbti_stats_for_radar
SELECT
      mbti.mbti_personality AS mbti_label,
      AVG(u.total_retweet_count) AS avg_retweets,
      AVG(u.total_favorite_count) AS avg_favorites,
      AVG(u.total_hashtag_count) AS avg_hashtags,
      AVG(u.total_mentions_count) AS avg_mentions,
      AVG(u.total_media_count) AS avg_media,
      AVG(fc.num_followers) AS avg_followers
FROM users u
JOIN mbti_labels mbti ON u.id = mbti.id
LEFT JOIN followers_count fc ON u.id = fc.user_id
GROUP BY mbti.mbti_personality;
```

Hau guztia egin ondoren, superset-en bistaraketa egin dezakegu. Superset hasieratzeko eta PostgreSQL datu iturri moduan gehitzeko pausuak lehen esplikatu direnez, ez dugu berriz egingo. Eta lehen esplikatu moduan **Datasets** atalean gehitu beharko dugu **mbti_stats_for_radar** sortu berri dugun taula (**PostgreSQL** eta **public** aukeratu baita).

Ondoren bistaraketaren konfigurazioa gauzatzeko dimensions eta metrics ataletan zer jarri hurrengo argazkiaren bidez ikus daiteke:


ROW LIMIT: 16 izango da

Bistaraketa hobeto ikusteko kontraste askoko koloreak erabiltzea gomendagarria da. Hori guztia azaldu ondoren ikus dezagun emaitza:




Grafiko honen helburua esan bezala, pertsonalitate bakoitzeko twitter-eko datu ezberdinen arteko erlazioa aztertzea da. Ideia da, mutur bakoitzean datu bat jarriz (adibidez retweet kopurua, jarraitzaile kopurua, mentzio kopurua…) pertsonalitate (kolore) bakoitzeko ikustea zer erlazio duten; kolore linea oso irekia bada eta mutur-muturretik mutur-muturrera badoa, esan nahiko du ezaugarri bat handia denean bestea ere handia izan ohi dela. Gainera, kolore guztiak batera ikustean, pertsonalitate ezberdinen kopuruak azter ditzakegu.

Adibidez, ikus dezakegu “favorite” kopurua oro har txikia dela pertsonalitate guztientzat, baina INTJ-rentzat handia egiten dela, nahiz eta retweet kopurua adibidez txikiagoa izan.

Hala ere, egia da ematen den informazioa batas bestekoa dela, eta honek benetan ez duela zertan adierazi erlazio hauen benetako naturaleza.

Bigarren bistaraketan erabiltzaile kopuruarekin ibiliko gara lanean. Zehazki zein letrak duen twiter-en ibiltzeko tendentzia gehiago edo gutxiago ikusiko dugu, baina binaka aztertuz. Hau da, extrovert vs introvert, sensing vs intuitive, feeling vs thinking eta judging vs perceiving. Horretarako box-plot-ak erabiliko ditugu. Lehen egin bezala lehenik PostgeSQL-n taulak sortu, ondoren Flink-en eta gero bistaraketa egingo dugu. mbti_labels eta followers_count berriz sortu beharko ditugu.

Beraz, PostgreSQL-n sortu beharrekoa:

```
docker exec -it postgres psql -U postgres -d mbti_analytics

CREATE TABLE mbti_dimensions (
      mbti_personality VARCHAR(10) PRIMARY KEY,
      EI VARCHAR(1),
      NS VARCHAR(1),
      FT VARCHAR(1),
      PJ VARCHAR(1),
      avg_followers DOUBLE PRECISION,
      num_users BIGINT
);
```

Eta ondoren flink-en:

```
docker exec -it jobmanager bash
/opt/flink/bin/sql-client.sh -l /opt/flink/lib

CREATE TABLE mbti_labels (
      id BIGINT,
      mbti_personality STRING,
      pers_id INT
) WITH (
      'connector' = 'jdbc',
      'url' = 'jdbc:mysql://mysql:3306/mbtidb',
      'table-name' = 'mbti_labels',
      'username' = 'root',
      'password' = 'root'
);

CREATE TABLE followers_count (
      user_id BIGINT,
```

```
      num_followers INT
) WITH (
      'connector' = 'jdbc',
      'url' = 'jdbc:mysql://mysql:3306/mbtidb',
      'table-name' = 'followers_count',
      'username' = 'root',
      'password' = 'root'
);

CREATE TABLE mbti_dimensions (
      mbti_personality STRING,
      EI STRING,
      NS STRING,
      FT STRING,
      PJ STRING,
      avg_followers DOUBLE,
      num_users BIGINT,
      PRIMARY KEY (mbti_personality) NOT ENFORCED
) WITH (
      'connector' = 'jdbc',
      'url' = 'jdbc:postgresql://postgres:5432/mbti_analytics',
      'table-name' = 'mbti_dimensions',
      'username' = 'postgres',
      'password' = 'postgres'
);

INSERT INTO mbti_dimensions
SELECT
      mbti_personality,
      LEFT(mbti_personality, 1) AS EI,
      SUBSTRING(mbti_personality, 2, 1) AS NS,
      SUBSTRING(mbti_personality, 3, 1) AS FT,
      RIGHT(mbti_personality, 1) AS PJ,
      AVG(f.num_followers) AS avg_followers,
      COUNT(*) AS num_users
FROM mbti_labels m
JOIN followers_count f ON m.id = f.user_id
GROUP BY mbti_personality;
```

Puntu honetan iadanik prest dugu guztia bistaraketak egiteko. Ideia da 4 bistaraketa egiteko, letrak bikoteka alderatzeko (ei, ns, ft, pj). Lehen egin dugun moduan **Datasets**-en berri bat gehitu: **mbti_dimensions**.

Jarraitu beharreko konfigurazioa (e vs i konparaketarako, adibidez):


Lortutako emaitzak ikusi ditzagun, lau konparaketetan, e vs i, n vs s, f vs t eta j vs p, hurrenez hurren:

Box-plot erabiliz letra bakoitzeko dagozkien 8 pertsonalitateak hartzen dira; adibidez ***i***-ren kasuan INTJ, INTP, INFJ, INFP, ISTJ, ISTP, ISFJ, ISFP. Horiekin kalkulatzen da box-plot-a. Beraz, mediana, kuantil eta outlierrak kontuan hartzen dira grafiko hauen bitartez. Orokorrean ikusten da alde handirik ez dagoela, asko solapatzen baitira. Baina, bigarren grafikoan ***s*** eta ***n*** alderatzen diren horretan, bai ikusten da ezberdintasun nabarmen bat, ***s*** direnak gutxiago dira, eta ez dago inolako solapaziorik. Hasiera batean, agian ***e*** eta ***i*** -ren

artean alde handiagoa egotea espero izan arre, grafikoak ikusi ondoren ikusi dugu oker geudela. Esan beharra dago, box-plot-etarako erabilitako datuak 16 pertsonalitateen batas bestekoak direla, eta agian interesgarriagoa litzatekeela, letra bakoitza duen erabiltzaile ororena banan banan hartzea pertsonalitateka multzokatzea baino. Hori etorkizunerako analisi moduan utzi dugu.

## 6. Amaiera

Proiektu honi esker, datu sozialen analisiaren inguruko ikuspegi praktiko bat lortu da, baita teknologia desberdinen arteko integrazioaren konplexutasunaz jabetzeko aukera ere. Sare sozialetako datuak egoki egituratu eta tratatuz gero, informazio aberatsa lor daiteke erabiltzaileen portaerari buruz, eta pertsonalitateak sareko jardueretan izan dezakeen eragina nabarmen daiteke zenbait kasutan. Hala ere, ondorio orokorrak ateratzerakoan kontuan hartu behar da erabilitako datu-bilduma mugatua dela, eta horrek emaitzen balio orokorra baldintzatu dezakeela. Edonola ere, lan honek erakutsi du datu sozialen azterketak aukera handiak eskaintzen dituela bai ikuspegi akademikotik bai aplikazio praktikoetatik, eta etorkizunean sakontzeko bide interesgarria dela.